Бесплатная электронная книга

УЧУСЬ

Microsoft SQL Server

Free unaffiliated eBook created from
Stack Overflow contributors.

#sql-server

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: microsoft-sql-server

It is an unofficial and free Microsoft SQL Server ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Microsoft SQL Server.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с Microsoft SQL Server

замечания

Это набор примеров, подчеркивающих базовое использование SQL Server.

Версии

Версия	Дата выхода
SQL Server 2016	2016-06-01
SQL Server 2014	2014-03-18
SQL Server 2012	2011-10-11
SQL Server 2008 R2	2010-04-01
SQL Server 2008	2008-08-06
SQL Server 2005	2005-11-01
SQL Server 2000	2000-11-01

Examples

INSERT / SELECT / UPDATE / DELETE: основы языка манипулирования данными

D ata **M** anipulation **L** anguage (короткий для DML) включает такие операции, как `INSERT` , `UPDATE` и `DELETE` :

```
-- Create a table HelloWorld

CREATE TABLE HelloWorld (
    Id INT IDENTITY,
    Description VARCHAR(1000)
)


-- DML Operation INSERT, inserting a row into the table
INSERT INTO HelloWorld (Description) VALUES ('Hello World')
```

```
-- DML Operation SELECT, displaying the table
SELECT * FROM HelloWorld


-- Select a specific column from table
SELECT Description FROM HelloWorld


-- Display number of records in the table
SELECT Count(*) FROM HelloWorld


-- DML Operation UPDATE, updating a specific row in the table
UPDATE HelloWorld SET Description = 'Hello, World!' WHERE Id = 1


-- Selecting rows from the table (see how the Description has changed after the update?)
SELECT * FROM HelloWorld


-- DML Operation - DELETE, deleting a row from the table
DELETE FROM HelloWorld WHERE Id = 1


-- Selecting the table. See table content after DELETE operation
SELECT * FROM HelloWorld
```

В этом скрипте мы **создаем таблицу** для демонстрации некоторых основных запросов.

В следующих примерах показано, как **запрашивать таблицы:**

```
USE Northwind;
GO
SELECT TOP 10 * FROM Customers
ORDER BY CompanyName
```

выберет первые 10 записей таблицы `Customer` , упорядоченные столбцом `CompanyName` из базы данных `Northwind` (которая является одной из примерных баз данных Microsoft, ее можно загрузить **отсюда**):

CustomerID	CompanyName	ContactName	ContactTitle	Address	City
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg
BOLID	Bólido Comidas preparadas	Martín Sommer	Owner	C/ Araquil, 67	Madrid
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen

Обратите внимание, что `Use Northwind;` изменяет базу данных по умолчанию для всех последующих запросов. Вы все равно можете ссылаться на базу данных, используя полный синтаксис в форме [Database]. [Схема]. [Таблица]:

```
SELECT TOP 10 * FROM Northwind.dbo.Customers
ORDER BY CompanyName

SELECT TOP 10 * FROM Pubs.dbo.Authors
ORDER BY City
```

Это полезно, если вы запрашиваете данные из разных баз данных. Обратите внимание, что `dbo` , заданный «in between», называется схемой и должен быть указан при использовании полного синтаксиса. Вы можете думать о нем как о папке в своей базе данных. `dbo` - это схема по умолчанию. Схема по умолчанию может быть опущена. Все остальные пользовательские схемы должны быть указаны.

Если таблица базы данных содержит столбцы, имена которых называются зарезервированными словами, например `Date` , вам нужно заключить имя столбца в скобки, например:

```
-- descending order
SELECT TOP 10 [Date] FROM dbo.MyLogTable
ORDER BY [Date] DESC
```

То же самое относится, если имя столбца содержит пробелы в имени (что не рекомендуется). Альтернативный синтаксис заключается в использовании двойных кавычек вместо квадратных скобок, например:

```
-- descending order
SELECT top 10 "Date" from dbo.MyLogTable
order by "Date" desc
```

эквивалентен, но не так часто используется. Обратите внимание на разницу между двойными кавычками и одинарными кавычками: одинарные кавычки используются для строк, т. Е.

```
-- descending order
SELECT top 10 "Date" from dbo.MyLogTable
where UserId='johndoe'
order by "Date" desc
```

является допустимым синтаксисом. Обратите внимание, что T-SQL имеет префикс `N` для типов данных NChar и NVarchar, например

```
SELECT TOP 10 * FROM Northwind.dbo.Customers
WHERE CompanyName LIKE N'AL%'
ORDER BY CompanyName
```

возвращает все компании, имеющие название компании, начиная с `AL` (`%` - это дикая карта, используйте ее так же, как вы бы использовали звездочку в командной строке DOS, например `DIR AL*`). Для `LIKE` существует несколько подстановочных знаков, посмотрите **здесь,** чтобы узнать подробности.

присоединяется

Соединения полезны, если вы хотите запрашивать поля, которые не существуют в одной таблице, но в нескольких таблицах. Например: вы хотите запросить все столбцы из таблицы `Region` в базе данных `Northwind` . Но вы заметили, что вам нужен также `RegionDescription` , который хранится в другой таблице `Region` . Тем не менее, существует общий ключ `RgionID` который можно использовать для объединения этой информации в один запрос следующим образом (`Top 5` просто возвращает первые 5 строк, опустите его, чтобы получить все строки):

```
SELECT TOP 5 Territories.*,
    Regions.RegionDescription
FROM Territories
INNER JOIN Region
    ON Territories.RegionID=Region.RegionID
ORDER BY TerritoryDescription
```

будут показаны все столбцы из `Territories` и столбец `RegionDescription` из `Region` . Результат упорядочен по `TerritoryDescription` .

Псевдонимы таблиц

Когда ваш запрос требует ссылки на две или более таблицы, вы можете счесть полезным использовать псевдоним таблицы. Табличные псевдонимы - это сокращенные ссылки на таблицы, которые можно использовать вместо полного имени таблицы, и могут уменьшить типизацию и редактирование. Синтаксис для использования псевдонима:

```
<TableName> [as] <alias>
```

Где `as` необязательное ключевое слово. Например, предыдущий запрос можно переписать как:

```
SELECT TOP 5 t.*,
    r.RegionDescription
FROM Territories t
INNER JOIN Region r
    ON t.RegionID = r.RegionID
ORDER BY TerritoryDescription
```

Псевдонимы должны быть уникальными для всех таблиц в запросе, даже если вы используете одну и ту же таблицу дважды. Например, если в таблице Employee указано поле SupervisorId, вы можете использовать этот запрос для возврата сотрудника и его имени руководителя:

```
SELECT e.*,
    s.Name as SupervisorName -- Rename the field for output
```

```
FROM Employee e
INNER JOIN Employee s
    ON e.SupervisorId = s.EmployeeId
WHERE e.EmployeeId = 111
```

Союзы

Как мы видели ранее, объединение добавляет столбцы из разных источников таблицы. Но что, если вы хотите объединить строки из разных источников? В этом случае вы можете использовать UNION. Предположим, вы планируете вечеринку и хотите пригласить не только сотрудников, но и клиентов. Затем вы можете запустить этот запрос, чтобы сделать это:

```
SELECT FirstName+' '+LastName as ContactName, Address, City FROM Employees
UNION
SELECT ContactName, Address, City FROM Customers
```

Он будет возвращать имена, адреса и города от сотрудников и клиентов в одной таблице. Обратите внимание, что дублирующиеся строки (если они должны быть какие-либо) автоматически удаляются (если вы этого не хотите, используйте вместо этого `UNION ALL`). Номер столбца, имена столбцов, порядок и тип данных должны соответствовать всем операторам выбора, которые являются частью объединения, поэтому первый SELECT объединяет `FirstName` и `LastName` от Employee в `ContactName` .

Переменные таблицы

Это может быть полезно, если вам нужно иметь дело с временными данными (особенно в хранимой процедуре), использовать табличные переменные: разница между «реальной» таблицей и табличной переменной заключается в том, что она существует только в памяти для временной обработки.

Пример:

```
DECLARE @Region TABLE
(
  RegionID int,
  RegionDescription NChar(50)
)
```

создает таблицу в памяти. В этом случае префикс `@` является обязательным, поскольку он является переменной. Вы можете выполнить все операции DML, упомянутые выше, для вставки, удаления и выбора строк, например

```
INSERT INTO @Region values(3,'Northern')
INSERT INTO @Region values(4,'Southern')
```

Но, как правило, вы заполняете его на основе реальной таблицы, например

```
INSERT INTO @Region
SELECT * FROM dbo.Region WHERE RegionID>2;
```

который будет считывать отфильтрованные значения из реальной таблицы `dbo.Region` и вставлять их в таблицу памяти `@Region` где ее можно использовать для дальнейшей обработки. Например, вы можете использовать его в

```
SELECT * FROM Territories t
JOIN @Region r on t.RegionID=r.RegionID
```

которая в этом случае вернет все `Northern` и `Southern` территории. Более подробную информацию можно найти здесь . Временные таблицы обсуждаются здесь , если вам интересно узнать больше об этой теме.

ПРИМЕЧАНИЕ. Microsoft рекомендует использовать переменные таблицы, если количество строк данных в переменной таблицы меньше 100. Если вы будете работать с большими объемами данных, вместо этого используйте **временную таблицу** или таблицу temp.

РАСПЕЧАТАТЬ

Отображение сообщения на выходной консоли. Используя SQL Server Management Studio, это будет отображаться на вкладке сообщений, а не на вкладке результатов:

```
PRINT 'Hello World!';
```

ВЫБРАТЬ все строки и столбцы из таблицы

Синтаксис:

```
SELECT *
FROM table_name
```

Использование оператора asterisk `*` служит ярлыком для выбора всех столбцов в таблице. Все строки также будут выбраны, потому что этот `SELECT` не имеет предложения `WHERE` , чтобы указать любые критерии фильтрации.

Это также будет работать так же, если вы добавили псевдоним в таблицу, например `e` в этом случае:

```
SELECT *
FROM Employees AS e
```

Или, если вы хотите выбрать все из конкретной таблицы, вы можете использовать

псевдоним + ". *":

```
SELECT e.*, d.DepartmentName
FROM Employees AS e
    INNER JOIN Department AS d
        ON e.DepartmentID = d.DepartmentID
```

К объектам базы данных также можно получить доступ с использованием полных имен:

```
SELECT * FROM [server_name].[database_name].[schema_name].[table_name]
```

Это не обязательно рекомендуется, так как изменение имени сервера и / или базы данных приведет к тому, что запросы с использованием полностью квалифицированных имен перестанут выполняться из-за неправильных имен объектов.

Обратите внимание, что поля before `table_name` могут быть опущены во многих случаях, если запросы выполняются на одном сервере, базе данных и схеме соответственно. Однако для базы данных часто существует несколько схем, и в этих случаях имя схемы не должно быть опущено, когда это возможно.

Предупреждение. Использование `SELECT *` в производственном коде или хранимых процедурах может привести к проблемам позже (при добавлении новых столбцов в таблицу или перестановке столбцов в таблице), особенно если ваш код делает простые предположения о порядке столбцов, или количество возвращенных столбцов. Поэтому безопаснее всегда явно указывать имена столбцов в операторах SELECT для производственного кода.

```
SELECT col1, col2, col3
FROM table_name
```

Выберите строки, соответствующие условию

Как правило, синтаксис:

```
SELECT <column names>
FROM <table name>
WHERE <condition>
```

Например:

```
SELECT FirstName, Age
FROM Users
WHERE LastName = 'Smith'
```

Условия могут быть сложными:

```
SELECT FirstName, Age
```

```
FROM Users
WHERE LastName = 'Smith' AND (City = 'New York' OR City = 'Los Angeles')
```

UPDATE Specific Row

```
UPDATE HelloWorlds
SET HelloWorld = 'HELLO WORLD!!!'
WHERE Id = 5
```

Вышеупомянутый код обновляет значение поля «HelloWorld» с «HELLO WORLD !!!» для записи, где «Id = 5» в таблице HelloWorlds.

Примечание. В операторе обновления рекомендуется использовать предложение «где», чтобы избежать обновления всей таблицы, пока и до тех пор, пока ваше требование не будет отличаться.

ОБНОВЛЕНИЕ ВСЕХ строк

Простая форма обновления увеличивает все значения в заданном поле таблицы. Для этого нам нужно определить поле и значение приращения

Ниже приведен пример, который увеличивает поле « `Score` на 1 (во всех строках):

```
UPDATE Scores
SET score = score + 1
```

Это может быть опасно, так как вы можете повредить свои данные, если вы случайно внесете UPDATE для **определенной строки** со **строками** UPDATE для **всех** в таблице.

Комментарии в коде

Transact-SQL поддерживает две формы написания комментариев. Комментарии игнорируются движком базы данных и предназначены для чтения людьми.

Комментарии предшествуют `--` и игнорируются до тех пор, пока не встретится новая строка:

```
-- This is a comment
SELECT *
FROM MyTable -- This is another comment
WHERE Id = 1;
```

Комментарии к слайду начинаются с `/*` и заканчиваются на `*/` . Весь текст между этими разделителями рассматривается как блок комментариев.

```
/* This is
a multi-line
```

```
comment block. */
SELECT Id = 1, [Message] = 'First row'
UNION ALL
SELECT 2, 'Second row'
/* This is a one liner */
SELECT 'More';
```

Замечания звезд Slash имеют то преимущество, что комментарий может использоваться, если SQL Statement теряет новые строковые символы. Это может произойти при захвате SQL во время устранения неполадок.

Комментарии к Slash star могут быть вложенными, а начало `/*` внутри комментария слэш-символом должно заканчиваться на `*/` чтобы быть действительным. Следующий код приведет к ошибке

```
/*
SELECT *
FROM CommentTable
WHERE Comment = '/*'
*/
```

Слэш-звезда, хотя внутри цитаты считается началом комментария. Следовательно, его нужно закончить с помощью другой закрывающей звезды. Правильный путь

```
/*
SELECT *
FROM CommentTable
WHERE Comment = '/*'
*/  */
```

Получить базовую информацию о сервере

```
SELECT @@VERSION
```

Возвращает версию MS SQL Server, запущенную на экземпляре.

```
SELECT @@SERVERNAME
```

Возвращает имя экземпляра MS SQL Server.

```
SELECT @@SERVICENAME
```

Возвращает имя службы Windows. MS SQL Server работает как.

```
SELECT serverproperty('ComputerNamePhysicalNetBIOS');
```

Возвращает физическое имя машины, на которой работает SQL Server. Полезно для идентификации узла в отказоустойчивом кластере.

```
SELECT * FROM fn_virtualservernodes();
```

В отказоустойчивом кластере возвращается каждый узел, на котором может работать SQL Server. Он не возвращает ничего, если не кластер.

Использование транзакций для безопасного изменения данных

Всякий раз, когда вы меняете данные, в команде Data Manipulation Language (DML) вы можете обернуть свои изменения в транзакции. DML включает `UPDATE` , `TRUNCATE` , `INSERT` и `DELETE` . Один из способов, которым вы можете убедиться, что вы меняете правильные данные, - это использовать транзакцию.

Изменения DML будут блокировать затронутые строки. Когда вы начинаете транзакцию, вы должны завершить транзакцию, или все объекты, которые будут изменены в DML, останутся заблокированными тем, кто начал транзакцию. Вы можете завершить транзакцию с помощью `ROLLBACK` или `COMMIT` . `ROLLBACK` возвращает все транзакции в исходное состояние. `COMMIT` помещает данные в конечное состояние, где вы не можете отменить свои изменения без другого оператора DML.

Пример:

```
--Create a test table

USE [your database]
GO
CREATE TABLE test_transaction (column_1 varchar(10))
GO

INSERT INTO
 dbo.test_transaction
        ( column_1 )
VALUES
        ( 'a' )

BEGIN TRANSACTION --This is the beginning of your transaction

UPDATE dbo.test_transaction
SET column_1 = 'B'
OUTPUT INSERTED.*
WHERE column_1 = 'A'


ROLLBACK TRANSACTION  --Rollback will undo your changes
           --Alternatively, use COMMIT to save your results

SELECT * FROM dbo.test_transaction   --View the table after your changes have been run

DROP TABLE dbo.test_transaction
```

Заметки:

- Это **упрощенный пример,** который не включает обработку ошибок. Но любая

операция базы данных может выйти из строя и, следовательно, вызвать исключение. **Вот пример** того, как может выглядеть такая необходимая обработка ошибок. Вы **никогда не** должны использовать транзакции **без обработчика ошибок** , иначе вы можете оставить транзакцию в неизвестном состоянии.

- В зависимости от **уровня изоляции** транзакции помещают блокировки на запрашиваемые или измененные данные. Вам необходимо убедиться, что транзакции не работают в течение длительного времени, поскольку они будут блокировать записи в базе данных и могут привести к **взаимоблокировкам** с другими параллельными транзакциями. Держите операции, заключенные в транзакции как можно короче, и минимизируйте влияние количества данных, которые вы блокируете.

УДАЛИТЬ ВСЕ строки

```
DELETE
FROM Helloworlds
```

Это приведет к удалению всех данных из таблицы. После запуска этого кода таблица не будет содержать строк. В отличие от `DROP TABLE` , это сохраняет таблицу и ее структуру, и вы можете продолжать вставлять новые строки в эту таблицу.

Другой способ удалить все строки в таблице - усечь его, как показано ниже:

```
TRUNCATE TABLE HelloWords
```

Разница с операцией DELETE несколько:

1. Операция Truncate не сохраняется в файле журнала транзакций
2. Если существует поле `IDENTITY` , это будет сброшено
3. TRUNCATE может применяться на всей таблице, а не на части (вместо этого с помощью команды `DELETE` вы можете связать предложение `WHERE`)

Ограничения TRUNCATE

1. Невозможно TRUNCATE таблицы, если есть ссылка `FOREIGN KEY`
2. Если в таблице участвуют `INDEXED VIEW`
3. Если таблица публикуется с использованием `TRANSACTIONAL REPLICATION` или `MERGE REPLICATION`
4. Он не будет запускать любой TRIGGER, определенный в таблице

[так в оригинале]

TRUNCATE TABLE

```
TRUNCATE TABLE Helloworlds
```

Этот код удалит все данные из таблицы Helloworlds. Таблица усечений почти аналогична `Delete from Table` . Разница в том, что вы не можете использовать where clauses с Truncate. Таблица усечения считается лучше, чем удаление, поскольку она использует меньше пространства журналов транзакций.

Обратите внимание, что если столбец идентификатора существует, он сбрасывается до начального начального значения (например, автоматически увеличиваемый ID будет перезапущен с 1). Это может привести к несогласованности, если столбцы идентификации используются в качестве внешнего ключа в другой таблице.

Создать новую таблицу и вставить записи из старой таблицы

```
SELECT * INTO NewTable FROM OldTable
```

Создает новую таблицу со структурой старой таблицы и вставляет все строки в новую таблицу.

Некоторые ограничения

1. Вы не можете указать в качестве новой таблицы переменную таблицы или таблицы.
2. Вы не можете использовать SELECT ... INTO для создания секционированной таблицы, даже если исходная таблица разделена. SELECT ... INTO не использует схему раздела исходной таблицы; вместо этого новая таблица создается в файловой группе по умолчанию. Чтобы вставлять строки в секционированную таблицу, вы должны сначала создать секционированную таблицу, а затем использовать инструкцию INSERT INTO ... SELECT FROM.
3. Индексы, ограничения и триггеры, определенные в исходной таблице, не переносятся в новую таблицу и не могут быть указаны в инструкции SELECT ... INTO. Если эти объекты необходимы, их можно создать после выполнения инструкции SELECT ... INTO.
4. Указание предложения ORDER BY не гарантирует, что строки будут вставлены в указанном порядке. Когда разреженный столбец включен в список выбора, свойство разреженного столбца не переносится в столбец в новой таблице. Если это свойство требуется в новой таблице, измените определение столбца после выполнения инструкции SELECT ... INTO, чтобы включить это свойство.
5. Когда вычисленный столбец включен в список выбора, соответствующий столбец в новой таблице не является вычисленным столбцом. Значения в новом столбце - это значения, которые были вычислены в момент выполнения SELECT ... INTO.

[*sic*]

Получение таблицы Row Count

Следующий пример может быть использован , чтобы найти общее количество строк для конкретной таблицы в базе данных , если `table_name` заменяется таблицами , которую вы хотите запросить:

```
SELECT COUNT(*) AS [TotalRowCount] FROM table_name;
```

Также можно получить подсчет строк для всех таблиц, присоединившись к разделу таблицы, основываясь на таблицах HEAP (index_id = 0) или кластерном индексе кластера (index_id = 1), используя следующий скрипт:

```
SELECT  [Tables].name                AS [TableName],
        SUM( [Partitions].[rows] )    AS [TotalRowCount]
FROM    sys.tables AS [Tables]
JOIN    sys.partitions AS [Partitions]
    ON  [Tables].[object_id]    =    [Partitions].[object_id]
    AND [Partitions].index_id IN ( 0, 1 )
--WHERE    [Tables].name = N'table name' /* uncomment to look for a specific table */
GROUP BY    [Tables].name;
```

Это возможно, так как каждая таблица по существу является отдельной таблицей разделов, если к ней не добавляются дополнительные разделы. Этот скрипт также имеет преимущество не вмешиваться в операции чтения / записи в строки таблиц.

Прочитайте Начало работы с Microsoft SQL Server онлайн: https://riptutorial.com/ru/sql-server/topic/236/начало-работы-с-microsoft-sql-server

глава 2: bcp (программа массовой копии) Утилита

Вступление

Объемная программа (bcp) для массовой копии копирует данные между экземпляром Microsoft SQL Server и файлом данных в указанном пользователем формате. Утилита bcp может использоваться для импорта большого количества новых строк в таблицы SQL Server или для экспорта данных из таблиц в файлы данных.

Examples

Пример импорта данных без файла формата (с использованием собственного формата)

```
REM Truncate table (for testing)
SQLCMD -Q "TRUNCATE TABLE TestDatabase.dbo.myNative;"

REM Import data
bcp TestDatabase.dbo.myNative IN D:\BCP\myNative.bcp -T -n

REM Review results
SQLCMD -Q "SELECT * FROM TestDatabase.dbo.myNative;"
```

Прочитайте bcp (программа массовой копии) Утилита онлайн: https://riptutorial.com/ru/sql-server/topic/10942/bcp--программа-массовой-копии--утилита

глава 3: COALESCE

Синтаксис

- COALESCE ([Column1], [Столбец2] .... [ColumnN]

Examples

Использование COALESCE для создания строкой с разделителями-запятыми

Мы можем получить строку с разделителями-запятыми из нескольких строк, используя coalesce, как показано ниже.

Поскольку используется переменная table, нам нужно выполнить весь запрос один раз. Поэтому, чтобы упростить понимание, я добавил блок BEGIN и END.

```
BEGIN

    --Table variable declaration to store sample records
    DECLARE @Table TABLE (FirstName varchar(256), LastName varchar(256))

    --Inserting sample records into table variable @Table
    INSERT INTO @Table (FirstName, LastName)
    VALUES
    ('John','Smith'),
    ('Jane','Doe')

    --Creating variable to store result
    DECLARE @Names varchar(4000)

    --Used COLESCE function, so it will concatenate comma seperated FirstName into @Names
varible
    SELECT @Names = COALESCE(@Names + ',', '') + FirstName
    FROM @Table

    --Now selecting actual result
    SELECT @Names
    END
```

Общий пример Coalesce

`COALESCE()` возвращает первое `NON NULL` в списке аргументов. Предположим, у нас была таблица, содержащая номера телефонов и номера сотовых телефонов, и мы хотели вернуть только один для каждого пользователя. Чтобы получить только один, мы можем получить первое `NON NULL` .

```
DECLARE @Table TABLE (UserID int, PhoneNumber varchar(12), CellNumber varchar(12))
```

```
INSERT INTO @Table (UserID, PhoneNumber, CellNumber)
VALUES
(1,'555-869-1123',NULL),
(2,'555-123-7415','555-846-7786'),
(3,NULL,'555-456-8521')


SELECT
    UserID,
    COALESCE(PhoneNumber, CellNumber)
FROM
    @Table
```

Получение первого значения не из списка значений столбца

```
SELECT COALESCE(NULL, NULL, 'TechOnTheNet.com', NULL, 'CheckYourMath.com');
Result: 'TechOnTheNet.com'

SELECT COALESCE(NULL, 'TechOnTheNet.com', 'CheckYourMath.com');
Result: 'TechOnTheNet.com'

SELECT COALESCE(NULL, NULL, 1, 2, 3, NULL, 4);
Result: 1
```

Прочитайте COALESCE онлайн: https://riptutorial.com/ru/sql-server/topic/3234/coalesce

глава 4: DBCC

Examples

Команды обслуживания DBCC

Команды DBCC позволяют пользователю поддерживать пространство в базе данных, очищать кеши, сокращать базы данных и таблицы.

Примерами являются:

```
DBCC DROPCLEANBUFFERS
```

Удаляет все чистые буферы из пула буферов и объекты столбцов из пула объектов столбца.

```
DBCC FREEPROCCACHE
-- or
DBCC FREEPROCCACHE (0x060006001ECA270EC0215D05000000000000000000000000);
```

Удаляет весь SQL-запрос в кеше плана. Каждый новый план будет перекомпилирован: вы можете указать дескриптор плана, обработчик запроса для очистки планов для конкретного плана запроса или оператора SQL.

```
DBCC FREESYSTEMCACHE ('ALL', myresourcepool);
-- or
DBCC FREESYSTEMCACHE;
```

Очищает все кэшированные записи, созданные системой. Он может очищать записи о = во всех или в определенном пуле ресурсов (**myresourcepool** в приведенном выше примере)

```
DBCC FLUSHAUTHCACHE
```

Опорожняет кеш аутентификации базы данных, содержащий информацию о логинах и правилах брандмауэра.

```
DBCC SHRINKDATABASE (MyDB [, 10]);
```

Сжимает базу данных MyDB до 10%. Второй параметр является необязательным. Вы можете использовать идентификатор базы данных вместо имени.

```
DBCC SHRINKFILE (DataFile1, 7);
```

Сжимает файл данных с именем DataFile1 в текущей базе данных. Размер цели - 7 МБ

(параметр tis не является обязательным).

```
DBCC CLEANTABLE (AdventureWorks2012,'Production.Document', 0)
```

Возвращает пробел из указанной таблицы

Утверждения валидации DBCC

Команды DBCC позволяют пользователю проверять состояние базы данных.

```
ALTER TABLE Table1 WITH NOCHECK ADD CONSTRAINT chkTab1 CHECK (Col1 > 100);
GO
DBCC CHECKCONSTRAINTS(Table1);
--OR
DBCC CHECKCONSTRAINTS ('Table1.chkTable1');
```

Проверить ограничение добавлено с параметрами nocheck, поэтому он не будет проверяться на существующих данных. DBCC вызовет проверку ограничений.

Следуя командам DBCC, проверьте целостность базы данных, таблицы или каталога:

```
DBCC CHECKTABLE tablename1 | tableid
DBCC CHECKDB databasename1 | dbid
DBCC CHECKFILEGROUP filegroup_name | filegroup_id | 0
DBCC CHECKCATALOG databasename1 | database_id1 | 0
```

Информационная инструкция DBCC

Команды DBCC могут показывать информацию о объектах базы данных.

```
DBCC PROCCACHE
```

Отображает информацию в формате таблицы о кеше процедуры.

```
DBCC OUTPUTBUFFER ( session_id [ , request_id ])
```

Возвращает текущий выходной буфер в шестнадцатеричном и ASCII формате для указанного session_id (и необязательный request_id).

```
DBCC INPUTBUFFER ( session_id [ , request_id ])
```

Отображает последний оператор, отправленный клиентом на экземпляр Microsoft SQL Server.

```
DBCC SHOW_STATISTICS ( table_or_indexed_view_name , column_statistic_or_index_name)
```

Команды трассировки DBCC

Флаги трассировки в SQL Server используются для изменения поведения SQL-сервера, включения / выключения некоторых функций. Команды DBCC могут контролировать флаги трассировки:

Следующий пример включает флаг трассировки 3205 в глобальном масштабе и 3206 для текущего сеанса:

```
DBCC TRACEON (3205, -1);
DBCC TRACEON (3206);
```

Следующий пример отключает флаг 3205 трассировки по всему миру и 3206 для текущего сеанса:

```
DBCC TRACEON (3205, -1);
DBCC TRACEON (3206);
```

В следующем примере отображается состояние флагов трассировки 2528 и 3205:

```
DBCC TRACESTATUS (2528, 3205);
```

Заявление DBCC

Заявления DBCC действуют как команды консоли базы данных для SQL Server. Чтобы получить синтаксическую информацию для указанной команды DBCC, используйте инструкцию DBCC HELP (...).

Следующий пример возвращает все инструкции DBCC, для которых доступна справка:

```
DBCC HELP ('?');
```

Следующий пример возвращает параметры для инструкции DBCC CHECKDB:

```
DBCC HELP ('CHECKDB');
```

Прочитайте DBCC онлайн: https://riptutorial.com/ru/sql-server/topic/7316/dbcc

глава 5: DBMAIL

Синтаксис

- sp_send_dbmail [[@profile_name =] 'profile_name'] [, [@recipients =] 'recipients [; ... n] '] [, [@copy_recipients =]' copy_recipient [; ... n] '] [, [@blind_copy_recipients =]' blind_copy_recipient [; ... n] '] [, [@from_address =]' from_address '] [, [@reply_to =]' reply_to '] [, [@subject =]' subject '] [, [@body =]' body '] [, [@body_format =] 'body_format'] [, [@importance =] 'важность'] [, [@sensitivity =] 'sensitive'] [, [@file_attachments =] 'attachment [; [n] '] [, [@query =]' query '] [, [@execute_query_database =]' execute_query_database '] [, [@attach_query_result_as_file =] attach_query_result_as_file] [, [@query_attachment_filename =] query_attachment_filename] [, [@query_result_header =] query_result_header] [, [@query_result_width =] query_result_width] [, [@query_result_separator =] 'query_result_separator'] [, [@exclude_query_output =] exclude_query_output] [, [@append_query_error =] append_query_error] [, [@query_no_truncate =] query_no_truncate] [, [@query_result_no_padding =] @query_result_no_padding] [, [@mailitem_id =] mailitem_id] [ВЫХОД]

Examples

Отправить простой адрес электронной почты

Этот код отправляет простое текстовое письмо по адресу recipient@someaddress.com

```
EXEC msdb.dbo.sp_send_dbmail
    @profile_name = 'The Profile Name',
    @recipients = 'recipient@someaddress.com',
    @body = 'This is a simple email sent from SQL Server.',
    @subject = 'Simple email'
```

Отправить результаты запроса

Это придает результаты запроса `SELECT * FROM Users` и отправляет его по `recipient@someaddress.com`

```
EXEC msdb.dbo.sp_send_dbmail
    @profile_name = 'The Profile Name',
    @recipients = 'recipient@someaddress.com',
    @query = 'SELECT * FROM Users',
    @subject = 'List of users',
    @attach_query_result_as_file = 1;
```

Отправить HTML-адрес электронной почты

Содержимое HTML должно быть передано в `sp_send_dbmail`

SQL Server 2012

```
DECLARE @html VARCHAR(MAX);
SET @html = CONCAT
(
    '<html><body>',
    '<h1>Some Header Text</h1>',
    '<p>Some paragraph text</p>',
    '</body></html>'
)
```

SQL Server 2012

```
DECLARE @html VARCHAR(MAX);
SET @html =
    '<html><body>' +
    '<h1>Some Header Text</h1>' +
    '<p>Some paragraph text</p>' +
    '</body></html>';
```

Затем используйте переменную `@html` с `@body argument` . Строка HTML также может быть передана непосредственно `@body` , хотя это может затруднить чтение кода.

```
EXEC msdb.dbo.sp_send_dbmail
    @recipients='recipient@someaddress.com',
    @subject = 'Some HTML content',
    @body = @html,
    @body_format = 'HTML';
```

Прочитайте DBMAIL онлайн: https://riptutorial.com/ru/sql-server/topic/4908/dbmail

глава 6: FileStream

Вступление

FILESTREAM объединяет SQL Server Database Engine с файловой системой NTFS, сохраняя данные двоичного большого двоичного файла (BLOB) в формате varbinary (max) в виде файлов в файловой системе. Операторы Transact-SQL могут вставлять, обновлять, запрашивать, искать и создавать резервные копии данных FILESTREAM. Интерфейсы файловой системы Win32 обеспечивают потоковый доступ к данным.

Examples

пример

Источник: MSDN https://technet.microsoft.com/en-us/library/bb933993(v=sql.105).aspx

Прочитайте FileStream онлайн: https://riptutorial.com/ru/sql-server/topic/9509/filestream

глава 7: FOR XML PATH

замечания

Существует также несколько других режимов `FOR XML` :

- `FOR XML RAW` - Создает один элемент `<row>` каждой строки.
- `FOR XML AUTO` - попытка эвристического автогенерации иерархии.
- `FOR XML EXPLICIT` - обеспечивает больший контроль над формой XML, но является более громоздким, чем `FOR XML PATH` .

Examples

Hello World XML

```
SELECT 'Hello World' FOR XML PATH('example')
```

```
<example>Hello World</example>
```

Указание пространств имен

SQL Server 2008

```
WITH XMLNAMESPACES (
    DEFAULT 'http://www.w3.org/2000/svg',
    'http://www.w3.org/1999/xlink' AS xlink
)
SELECT
    'example.jpg' AS 'image/@xlink:href',
    '50px' AS 'image/@width',
    '50px' AS 'image/@height'
FOR XML PATH('svg')
```

```
<svg xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://www.w3.org/2000/svg">
    <image xlink:href="firefox.jpg" width="50px" height="50px"/>
</svg>
```

Определение структуры с использованием выражений XPath

```
SELECT
    'XPath example' AS 'head/title',
    'This example demonstrates ' AS 'body/p',
    'https://www.w3.org/TR/xpath/' AS 'body/p/a/@href',
    'XPath expressions' AS 'body/p/a'
FOR XML PATH('html')
```

```
<html>
    <head>
        <title>XPath example</title>
    </head>
    <body>
        <p>This example demonstrates <a href="https://www.w3.org/TR/xpath/">XPath
expressions</a></p>
    </body>
</html>
```

В `FOR XML PATH` столбцы без имени становятся текстовыми узлами. `NULL` или `''` поэтому становятся пустыми текстовыми узлами. Примечание. Вы можете преобразовать именованный столбец в неназванный, используя `AS *`

```
DECLARE @tempTable TABLE (Ref INT, Des NVARCHAR(100), Qty INT)
INSERT INTO @tempTable VALUES (100001, 'Normal', 1), (100002, 'Foobar', 1), (100003, 'Hello
World', 2)

SELECT ROW_NUMBER() OVER (ORDER BY Ref) AS '@NUM',
     'REF' AS 'FLD/@NAME', REF AS 'FLD', '',
     'DES' AS 'FLD/@NAME', DES AS 'FLD', '',
     'QTY' AS 'FLD/@NAME', QTY AS 'FLD'
FROM @tempTable
FOR XML PATH('LIN'), ROOT('row')
```

```
<row>
  <LIN NUM="1">
    <FLD NAME="REF">100001</FLD>
    <FLD NAME="DES">Normal</FLD>
    <FLD NAME="QTY">1</FLD>
  </LIN>
  <LIN NUM="2">
    <FLD NAME="REF">100002</FLD>
    <FLD NAME="DES">Foobar</FLD>
    <FLD NAME="QTY">1</FLD>
  </LIN>
  <LIN NUM="3">
    <FLD NAME="REF">100003</FLD>
    <FLD NAME="DES">Hello World</FLD>
    <FLD NAME="QTY">2</FLD>
  </LIN>
</row>
```

Использование (пустых) текстовых узлов помогает отделить предыдущий выходной узел от следующего, так что SQL Server знает, как запустить новый элемент для следующего столбца. В противном случае он запутывается, когда атрибут уже существует в том, что он считает «текущим» элементом.

Например, без пустых строк между элементом и атрибутом в `SELECT` SQL Server дает ошибку:

> Столбец с атрибутом «FLD / @ NAME» не должен появляться после не-атрибут-центричного брата в иерархии XML в FOR XML PATH.

Также обратите внимание, что этот пример также обернул XML в корневой элемент с именем `row` , указанный `ROOT('row')`

Использование FOR XML PATH для конкатенации значений

`FOR XML PATH` можно использовать для конкатенации значений в строку. Следующий пример объединяет значения в строку `CSV` :

```
DECLARE @DataSource TABLE
(
    [rowID] TINYINT
   ,[FirstName] NVARCHAR(32)
);

INSERT INTO @DataSource ([rowID], [FirstName])
VALUES (1, 'Alex')
      ,(2, 'Peter')
      ,(3, 'Alexsandyr')
      ,(4, 'George');

SELECT STUFF
(
    (
        SELECT ',' + [FirstName]
        FROM @DataSource
        ORDER BY [rowID] DESC
        FOR XML PATH(''), TYPE
    ).value('.', 'NVARCHAR(MAX)')
    ,1
    ,1
    ,''
);
```

Несколько важных замечаний:

- предложение `ORDER BY` может использоваться для упорядочения значений в предпочтительном порядке
- если в качестве разделителя конкатенации используется более длинное значение, необходимо также изменить параметр функции `STUFF` ;

```
SELECT STUFF
(
    (
        SELECT '---' + [FirstName]
        FROM @DataSource
        ORDER BY [rowID] DESC
        FOR XML PATH(''), TYPE
    ).value('.', 'NVARCHAR(MAX)')
    ,1
    ,3 -- the "3" could also be represented as: LEN('---') for clarity
    ,''
);
```

- поскольку используются опция `TYPE` и `.value` , конкатенация работает с `NVARCHAR(MAX)`

Прочитайте FOR XML PATH онлайн: https://riptutorial.com/ru/sql-server/topic/727/for-xml-path

глава 8: JSON на сервере Sql

Синтаксис

- **JSON_VALUE** (выражение, путь) - извлекает скалярное значение из строки JSON.
- **JSON_QUERY** (выражение [, путь]) - извлекает объект или массив из строки JSON.
- **OPENJSON** (jsonExpression [, path]) - функция table-value, которая анализирует текст JSON и возвращает объекты и свойства в JSON в виде строк и столбцов.
- **ISJSON** (выражение) - проверяет, содержит ли строка действительный JSON.
- **JSON_MODIFY** (выражение, путь, newValue) - обновляет значение свойства в строке JSON и возвращает обновленную строку JSON.

параметры

параметры	подробности
выражение	Как правило, имя переменной или столбца, содержащего текст JSON.
дорожка	Выражение пути JSON, которое указывает свойство для обновления. path имеет следующий синтаксис: [append] [lax | strict] $. <json path>
jsonExpression	Является символьным символом Unicode, содержащим текст JSON.

замечания

Функция OPENJSON доступна только на уровне совместимости 130. Если уровень совместимости базы данных ниже 130, SQL Server не сможет найти и выполнить функцию OPENJSON. В настоящее время все базы данных Azure SQL по умолчанию установлены в 120. Вы можете изменить уровень совместимости базы данных, используя следующую команду:

```
ALTER DATABASE <Database-Name-Here> SET COMPATIBILITY_LEVEL = 130
```

Examples

Форматировать результаты запроса как JSON с FOR JSON

Данные входных таблиц (таблица «Люди»)

Я бы	название	Возраст
1	Джон	23
2	Джейн	31

запрос

```
SELECT Id, Name, Age
FROM People
FOR JSON PATH
```

Результат

```
[
    {"Id":1,"Name":"John","Age":23},
    {"Id":2,"Name":"Jane","Age":31}
]
```

Текст анализа JSON

Функции JSON_VALUE и **JSON_QUERY** анализируют текст JSON и возвращают скалярные значения или объекты / массивы на пути в тексте JSON.

```
DECLARE @json NVARCHAR(100) = '{"id": 1, "user":{"name":"John"}, "skills":["C#","SQL"]}'

SELECT
    JSON_VALUE(@json, '$.id') AS Id,
    JSON_VALUE(@json, '$.user.name') AS Name,
    JSON_QUERY(@json, '$.user') AS UserObject,
    JSON_QUERY(@json, '$.skills') AS Skills,
    JSON_VALUE(@json, '$.skills[0]') AS Skill0
```

Результат

Я бы	название	UserObject	Навыки	Skill0
1	Джон	{ "Имя": "Джон"}	["C #", "SQL"]	C #

Присоединяйте родительские и дочерние объекты JSON, используя CROSS APPLY OPENJSON

Присоединяйте родительские объекты со своими дочерними объектами, например, мы хотим, чтобы реляционная таблица каждого человека и их хобби

```
DECLARE @json nvarchar(1000) =
N'[
    {
        "id":1,
```

```
        "user":{"name":"John"},
        "hobbies":[
            {"name": "Reading"},
            {"name": "Surfing"}
        ]
    },
    {
        "id":2,
        "user":{"name":"Jane"},
        "hobbies":[
            {"name": "Programming"},
            {"name": "Running"}
        ]
    }
 ]'
```

запрос

```
SELECT
    JSON_VALUE(person.value, '$.id') as Id,
    JSON_VALUE(person.value, '$.user.name') as PersonName,
    JSON_VALUE(hobbies.value, '$.name') as Hobby
FROM OPENJSON (@json) as person
    CROSS APPLY OPENJSON(person.value, '$.hobbies') as hobbies
```

В качестве альтернативы этот запрос можно записать с помощью предложения WITH.

```
SELECT
    Id, person.PersonName, Hobby
FROM OPENJSON (@json)
WITH(
    Id int '$.id',
    PersonName nvarchar(100) '$.user.name',
    Hobbies nvarchar(max) '$.hobbies' AS JSON
) as person
CROSS APPLY OPENJSON(Hobbies)
WITH(
    Hobby nvarchar(100) '$.name'
)
```

Результат

Я бы	PersonName	Хобби
1	Джон	чтение
1	Джон	серфинг
2	Джейн	программирование
2	Джейн	Бег

Индекс по свойствам JSON с использованием вычисленных столбцов

При хранении JSON-документов в SQL Server нам необходимо иметь возможность эффективно фильтровать и сортировать результаты запроса по свойствам документов JSON.

```
CREATE TABLE JsonTable
(
    id int identity primary key,
    jsonInfo nvarchar(max),
    CONSTRAINT [Content should be formatted as JSON]
    CHECK (ISJSON(jsonInfo)>0)
)
```

```
INSERT INTO JsonTable
VALUES(N'{"Name":"John","Age":23}'),
(N'{"Name":"Jane","Age":31}'),
(N'{"Name":"Bob","Age":37}'),
(N'{"Name":"Adam","Age":65}')
GO
```

Учитывая приведенную выше таблицу. Если мы хотим найти строку с именем = 'Adam', мы выполним следующий запрос.

```
SELECT *
FROM JsonTable Where
JSON_VALUE(jsonInfo, '$.Name') = 'Adam'
```

Однако для этого требуется, чтобы SQL-сервер выполнил полную таблицу, которая на большой таблице неэффективна.

Чтобы ускорить это, мы хотели бы добавить индекс, однако мы не можем напрямую ссылаться на свойства в документе JSON. Решение состоит в том, чтобы добавить вычисленный столбец в пути JSON `$.Name` , а затем добавить индекс в вычисленный столбец.

```
ALTER TABLE JsonTable
ADD vName as JSON_VALUE(jsonInfo, '$.Name')

CREATE INDEX idx_name
ON JsonTable(vName)
```

Теперь, когда мы выполняем один и тот же запрос, вместо полного сканирования таблицы SQL-сервер использует индекс для поиска в некластеризованном индексе и поиск строк, удовлетворяющих заданным условиям.

Примечание. Чтобы SQL-сервер использовал индекс, вы должны создать вычисленный столбец с тем же выражением, которое вы планируете использовать в своих запросах, - в этом примере `JSON_VALUE(jsonInfo, '$.Name')` , однако вы также можете использовать имя вычисленного столбца `vName`

Отформатируйте одну строку таблицы как единый объект JSON, используя FOR JSON

Параметр WITHOUT_ARRAY_WRAPPER в предложении *FOR JSON* удаляет скобки массива с выхода JSON. Это полезно, если вы возвращаете одну строку в запросе.

> Примечание. Эта опция выдаст недопустимый вывод JSON, если возвращается более одной строки.

Данные входных таблиц (таблица «Люди»)

Я бы	название	Возраст
1	Джон	23
2	Джейн	31

запрос

```
SELECT Id, Name, Age
FROM People
WHERE Id = 1
FOR JSON PATH, WITHOUT_ARRAY_WRAPPER
```

Результат

```
{"Id":1,"Name":"John","Age":23}
```

Разбирать текст JSON с использованием функции OPENJSON

Функция **OPENJSON** анализирует текст JSON и возвращает несколько выходов. Значения, которые должны быть возвращены, указываются с использованием путей, определенных в предложении WITH. Если для некоторого столбца не указан путь, имя столбца используется как путь. Эта функция возвращает возвращаемые значения в типы SQL, определенные в предложении WITH. Опция AS JSON должна указываться в определении столбца, если какой-либо объект / массив должен быть возвращен.

```
DECLARE @json NVARCHAR(100) = '{"id": 1, "user":{"name":"John"}, "skills":["C#","SQL"]}'

SELECT *
FROM OPENJSON (@json)
    WITH(Id int '$.id',
        Name nvarchar(100) '$.user.name',
        UserObject nvarchar(max) '$.user' AS JSON,
        Skills nvarchar(max) '$.skills' AS JSON,
        Skill0 nvarchar(20) '$.skills[0]')
```

Результат

Я бы	название	UserObject	Навыки	Skill0
1	Джон	{ "Имя": "Джон"}	["C #", "SQL"]	C #

Прочитайте JSON на сервере Sql онлайн: https://riptutorial.com/ru/sql-server/topic/2568/json-на-сервере-sql

глава 9: MERGE

Вступление

Начиная с SQL Server 2008, можно выполнять операции вставки, обновления или удаления в одном операторе, используя оператор MERGE.

Оператор MERGE позволяет вам присоединиться к источнику данных с целевой таблицей или представлением, а затем выполнить несколько действий против цели на основе результатов этого соединения.

Синтаксис

- В соответствии с MSDN - https://msdn.microsoft.com/en-us/library/bb510625.aspx [WITH <common_table_expression> [, ... n]] MERGE [TOP (выражение) [PERCENT]] [INTO] < target_table> [WITH (<merge_hint>)] [[AS] table_alias] ИСПОЛЬЗОВАНИЕ <table_source> ON <merge_search_condition> [КОГДА СООТВЕТСТВУЕТ [AND <clause_search_condition>] THEN <merge_matched>] [... n] [КОГДА НЕ СООТВЕТСТВУЕТ [BY TARGET] [AND <clause_search_condition>] THEN <merge_not_matched>] [КОГДА НЕ СООТВЕТСТВУЕТ ИСТОЧНИКОМ [AND <clause_search_condition>] THEN <merge_matched>] [... n] [<output_clause>] [OPTION (<query_hint> [,. ..n])]; <target_table> :: = {[имя_базы. имя_схемы. | имя_схемы.] target_table} <merge_hint> :: = {{[<table_hint_limited> [, ... n]] [[,] INDEX (index_val [, ... n])]}} <table_source> :: = {table_or_view_name [[AS] table_alias] [<tablesample_clause>] [WITH (table_hint [[,] ... n])] | rowset_function [[AS] table_alias] [(bulk_column_alias [, ... n])] | user_defined_function [[AS] table_alias] | OPENXML <openxml_clause> | found_table [AS] table_alias [(column_alias [, ... n])] | <join_table> | <pivoted_table> | <unpivoted_table>} <merge_search_condition> :: = <search_condition> <merge_matched> :: = {UPDATE SET <set_clause> | DELETE} <set_clause> :: = SET {column_name = {выражение | ПО УМОЛЧАНИЮ | NULL} | {udt_column_name. {{property_name = выражение | field_name = выражение} | method_name (argument [, ... n])}} | column_name {.WRITE (выражение, @ Offset, @Length)} | @variable = выражение | @variable = column = выражение | column_name {+ = | - = | * = | / = | % = | & = | ^ = | | =} выражение | @variable {+ = | - = | * = | / = | % = | & = | ^ = | | =} выражение | @variable = column {+ = | - = | * = | / = | % = | & = | ^ = | | =} выражение} [, ... n] <merge_not_matched> :: = {INSERT [(column_list)] {VALUES (values_list) | DEFAULT VALUES}} <clause_search_condition> :: = <search_condition> :: = {[NOT] | (<search_condition>)} [{AND | ИЛИ} [NOT] {| (<search_condition>)}] [, ... n] :: = { выражение {= | <> | ! = |

  | > = | ! > | <| <= | ! <} выражение | string_expression [NOT] LIKE string_expression [ESCAPE 'escape_character'] | выражение [NOT] BETWEEN выражение AND выражение | выражение IS [NOT] NULL | CONTAINS

({column | *}, '<contains_search_condition>') | FREETEXT ({column | *}, 'freetext_string') | выражение [NOT] IN (подзапрос | выражение [, ... n]) | выражение {= | <> | ! = | | > = | ! > | <| <= | ! <} {ВСЕ | НЕКОТОРЫЕ | ANY} (подзапрос) | EXISTS (подзапрос)} <output_clause> :: = {[OUTPUT <dml_select_list> INTO {@table_variable | output_table} [(column_list)]] [OUTPUT <dml_select_list>]} <dml_select_list> :: = {<column_name> | scalar_expression} [[AS] column_alias_identifier] [, ... n] <column_name> :: = {DELETED | ВСТАВЬТЕ | from_table_name}. {* | column_name} | $ действие

замечания

Выполняет операции вставки, обновления или удаления в целевой таблице на основе результатов соединения с исходной таблицей. Например, вы можете синхронизировать две таблицы, вставляя, обновляя или удаляя строки в одной таблице на основе различий, найденных в другой таблице.

Examples

MERGE для ввода / обновления / удаления

```
MERGE INTO targetTable

USING sourceTable
ON (targetTable.PKID = sourceTable.PKID)

WHEN MATCHED AND (targetTable.PKID > 100) THEN
    DELETE

WHEN MATCHED AND (targetTable.PKID <= 100) THEN
    UPDATE SET
        targetTable.ColumnA = sourceTable.ColumnA,
        targetTable.ColumnB = sourceTable.ColumnB

WHEN NOT MATCHED THEN
    INSERT (ColumnA, ColumnB) VALUES (sourceTable.ColumnA, sourceTable.ColumnB);

WHEN NOT MATCHED BY SOURCE THEN
    DELETE
; --< Required
```

Описание:

- `MERGE INTO targetTable` - таблица, подлежащая изменению
- `USING sourceTable` - источник данных (может быть функцией таблицы или представления или таблицы)
- `ON ...` - условие соединения между `targetTable` и `sourceTable` .
- `WHEN MATCHED` - действия, которые нужно предпринять, когда найден матч
- - `AND (targetTable.PKID > 100)` - дополнительное условие (условия), которое должно

быть выполнено для того, чтобы действие было предпринято

- `THEN DELETE` - удалить сопоставленную запись из `targetTable`
- `THEN UPDATE` - обновить столбцы согласованной записи, заданной `SET ...`.
- `WHEN NOT MATCHED` - действия, которые нужно предпринять, когда совпадение не найдено в **`targetTable`**
- `WHEN NOT MATCHED BY SOURCE` - действия, которые нужно предпринять, когда совпадение не найдено в **`sourceTable`**

Комментарии:

Если конкретное действие не требуется, `WHEN NOT MATCHED THEN INSERT` условие, например, удаляя `WHEN NOT MATCHED THEN INSERT` предотвратит `WHEN NOT MATCHED THEN INSERT` записей

Оператор слияния требует конечной точки с запятой.

Ограничения:

- `WHEN MATCHED` не разрешает действие `INSERT`
- Действие `UPDATE` может обновлять строку только один раз. Это означает, что условие объединения должно давать уникальные совпадения.

Слияние с использованием источника CTE

```
WITH SourceTableCTE AS
(
    SELECT * FROM SourceTable
)
MERGE
 TargetTable AS target
USING SourceTableCTE AS source
ON (target.PKID = source.PKID)
WHEN MATCHED THEN
    UPDATE SET target.ColumnA = source.ColumnA
WHEN NOT MATCHED THEN
    INSERT (ColumnA) VALUES (Source.ColumnA);
```

MERGE с использованием таблицы исходных источников

```
MERGE INTO TargetTable  AS Target
USING (VALUES (1,'Value1'), (2, 'Value2'), (3,'Value3'))
       AS Source (PKID, ColumnA)
ON Target.PKID = Source.PKID
WHEN MATCHED THEN
    UPDATE SET target.ColumnA= source.ColumnA
WHEN NOT MATCHED THEN
    INSERT (PKID, ColumnA) VALUES (Source.PKID, Source.ColumnA);
```

Пример объединения - Синхронизация исходной и целевой таблицы

Чтобы проиллюстрировать заявление MERGE, рассмотрите следующие две таблицы:

1. **dbo.Product** : эта таблица содержит информацию о продукте, который компания продает в настоящее время

2. **dbo.ProductNew** : эта таблица содержит информацию о продукте, который компания будет продавать в будущем.

Следующий T-SQL будет создавать и заполнять эти две таблицы

```
IF OBJECT_id(N'dbo.Product',N'U') IS NOT NULL
DROP TABLE dbo.Product
GO

CREATE TABLE dbo.Product (
ProductID INT PRIMARY KEY,
ProductName NVARCHAR(64),
PRICE MONEY
)

IF OBJECT_id(N'dbo.ProductNew',N'U') IS NOT NULL
DROP TABLE dbo.ProductNew
GO

CREATE TABLE dbo.ProductNew (
ProductID INT PRIMARY KEY,
ProductName NVARCHAR(64),
PRICE MONEY
)

INSERT INTO dbo.Product VALUES(1,'IPod',300)
,(2,'IPhone',400)
,(3,'ChromeCast',100)
,(4,'raspberry pi',50)

INSERT INTO dbo.ProductNew VALUES(1,'Asus Notebook',300)
,(2,'Hp Notebook',400)
,(3,'Dell Notebook',100)
,(4,'raspberry pi',50)
```

Теперь предположим, что мы хотим синхронизировать таблицу результатов dbo.Product с таблицей dbo.ProductNew. Вот критерий для этой задачи:

1. Продукт, который существует как в исходной таблице dbo.ProductNew, так и в целевой таблице dbo.Product, обновляется в целевой таблице dbo.Product с новыми новыми продуктами.

2. Любой продукт в dbo.ProductНовая исходная таблица, которая не существует в целевой таблице dob.Product, вставляется в целевую таблицу dbo.Product.

3. Любой продукт в целевой таблице dbo.Product, которая не существует в исходной таблице dbo.ProductNew, должна быть удалена из целевой таблицы dbo.Product. Ниже приведен оператор MERGE для выполнения этой задачи.

```
MERGE dbo.Product AS SourceTbl
```

```
USING dbo.ProductNew AS TargetTbl ON (SourceTbl.ProductID = TargetTbl.ProductID)
WHEN MATCHED
            AND SourceTbl.ProductName <> TargetTbl.ProductName
            OR SourceTbl.Price <> TargetTbl.Price
    THEN UPDATE SET SourceTbl.ProductName = TargetTbl.ProductName,
                SourceTbl.Price = TargetTbl.Price
WHEN NOT MATCHED
    THEN INSERT (ProductID, ProductName, Price)
         VALUES (TargetTbl.ProductID, TargetTbl.ProductName, TargetTbl.Price)
WHEN NOT MATCHED BY SOURCE
    THEN DELETE
OUTPUT $action, INSERTED.*, DELETED.*;
```

Примечание: точка с запятой должна присутствовать в конце инструкции MERGE.

	$action	ProductID	ProductName	PRICE	ProductID	ProductName	PRICE
1	UPDATE	1	Asus Notebook	300.00	1	IPod	300.00
2	UPDATE	2	Hp Notebook	400.00	2	IPhone	400.00
3	UPDATE	3	Dell Notebook	100.00	3	ChromeCast	100.00

Слияние с использованием EXCEPT

Используйте EXCEPT для предотвращения обновлений неизмененных записей

```
MERGE TargetTable targ
USING SourceTable AS src
    ON src.id = targ.id
WHEN MATCHED
    AND EXISTS (
        SELECT src.field
        EXCEPT
        SELECT targ.field
        )
    THEN
        UPDATE
        SET field = src.field
WHEN NOT MATCHED BY TARGET
    THEN
        INSERT (
            id
            ,field
            )
        VALUES (
            src.id
            ,src.field
            )
WHEN NOT MATCHED BY SOURCE
    THEN
        DELETE;
```

Прочитайте MERGE онлайн: https://riptutorial.com/ru/sql-server/topic/4550/merge

глава 10: OPENJSON

Examples

Получить ключ: пары значений из текста JSON

Функция OPENJSON анализирует текст JSON и возвращает все пары ключ: значение на первом уровне JSON:

```
declare @json NVARCHAR(4000) = N'{"Name":"Joe","age":27,"skills":["C#","SQL"]}';
SELECT * FROM OPENJSON(@json);
```

ключ	значение	тип
название	Джо	1
возраст	27	2
навыки	["C #", "SQL"]	4

Тип столбца описывает тип значения, то есть null (0), строку (1), число (2), логическое (3), массив (4) и объект (5).

Преобразование массива JSON в множество строк

Функция OPENJSON анализирует набор объектов JSON и возвращает значения из текста JSON в виде набора строк.

```
declare @json nvarchar(4000) = N'[
  {"Number":"SO43659","Date":"2011-05-31T00:00:00","Customer":
"MSFT","Price":59.99,"Quantity":1},
  {"Number":"SO43661","Date":"2011-06-
01T00:00:00","Customer":"Nokia","Price":24.99,"Quantity":3}
]'

SELECT    *
FROM OPENJSON (@json)
    WITH (
          Number   varchar(200),
          Date     datetime,
          Customer varchar(200),
          Quantity int
  )
```

В предложении WITH указывается схема возврата функции OPENJSON. Ключи в объектах JSON извлекаются по именам столбцов. Если какой-либо ключ в JSON не указан в предложении WITH (например, цена в этом примере), он будет проигнорирован. Значения

автоматически преобразуются в определенные типы.

Число	Дата	Покупатель	Количество
SO43659	2011-05-31T00: 00: 00	MSFT	1
SO43661	2011-06-01T00: 00: 00	Nokia	3

Преобразование вложенных полей JSON в набор строк

Функция OPENJSON анализирует набор объектов JSON и возвращает значения из текста JSON в виде набора строк. Если значения в входном объекте вложены, в каждом столбце в предложении WITH может быть указан дополнительный параметр сопоставления:

```
declare @json nvarchar(4000) = N'[
  {"data":{"num":"SO43659","date":"2011-05-
31T00:00:00"},"info":{"customer":"MSFT","Price":59.99,"qty":1}},
  {"data":{"number":"SO43661","date":"2011-06-
01T00:00:00"},"info":{"customer":"Nokia","Price":24.99,"qty":3}}
]'

SELECT    *
FROM OPENJSON (@json)
    WITH (
          Number   varchar(200) '$.data.num',
          Date     datetime '$.data.date',
          Customer varchar(200) '$.info.customer',
          Quantity int '$.info.qty',
  )
```

В предложении WITH указывается схема возврата функции OPENJSON. После того, как тип указан путь к узлам JSON, где должно быть найдено возвращаемое значение. Ключи в объектах JSON извлекаются этими путями. Значения автоматически преобразуются в определенные типы.

Число	Дата	Покупатель	Количество
SO43659	2011-05-31T00: 00: 00	MSFT	1
SO43661	2011-06-01T00: 00: 00	Nokia	3

Извлечение внутренних подсайтов JSON

OPENJSON может извлекать фрагменты объектов JSON внутри текста JSON. В определении столбца, которое ссылается на под-объект JSON, задается опция nvarchar (max) и AS JSON:

```
declare @json nvarchar(4000) = N'[
```

```
  {"Number":"SO43659","Date":"2011-05-
31T00:00:00","info":{"customer":"MSFT","Price":59.99,"qty":1}},
  {"Number":"SO43661","Date":"2011-06-
01T00:00:00","info":{"customer":"Nokia","Price":24.99,"qty":3}}
]'

SELECT    *
FROM OPENJSON (@json)
    WITH (
          Number   varchar(200),
          Date     datetime,
          Info nvarchar(max) '$.info' AS JSON
  )
```

Информационный столбец будет сопоставлен с объектом «Информация». Результатом будет:

Число	Дата	Информация
SO43659	2011-05-31T00: 00: 00	{ "Клиент": "MSFT", "Цена": 59,99, "кол-во": 1}
SO43661	2011-06-01T00: 00: 00	{ "Клиент": "Nokia", "Цена": 24,99, "кол-во": 3}

Работа с вложенными подматрицами JSON

JSON может иметь сложную структуру с внутренними массивами. В этом примере у нас есть массив заказов с вложенным под массивом OrderItems.

```
declare @json nvarchar(4000) = N'[
  {"Number":"SO43659","Date":"2011-05-31T00:00:00",
    "Items":[{"Price":11.99,"Quantity":1},{"Price":12.99,"Quantity":5}]},
  {"Number":"SO43661","Date":"2011-06-01T00:00:00",

"Items":[{"Price":21.99,"Quantity":3},{"Price":22.99,"Quantity":2},{"Price":23.99,"Quantity":2}]}

]'
```

Мы можем проанализировать свойства корневого уровня с помощью OPENJSON, который будет возвращать массив элементов AS JSON. Затем мы можем снова применить OPENJSON в массиве Items и открыть внутреннюю таблицу JSON. Таблица первого уровня и внутренняя таблица будут «объединены», как в JOIN между стандартными таблицами:

```
SELECT    *
FROM
    OPENJSON (@json)
    WITH (  Number varchar(200), Date datetime,
            Items nvarchar(max) AS JSON )
        CROSS APPLY
            OPENJSON (Items)
            WITH ( Price float, Quantity int)
```

Результаты:

Число	Дата	Предметы	Цена	Количество
SO43659	2011-05-31 00: 00: 00.000	[{ "Цена": 11,99, "Количество": 1}, { "Цена": 12,99, "Количество": 5}]	11,99	1
SO43659	2011-05-31 00: 00: 00.000	[{ "Цена": 11,99, "Количество": 1}, { "Цена": 12,99, "Количество": 5}]	12,99	5
SO43661	2011-06-01 00: 00: 00.000	[{ "Цена": 21.99, "Количество": 3}, { "Цена": 22.99, "Количество": 2}, { "Цена": 23.99, "Количество": 2}]	21,99	3
SO43661	2011-06-01 00: 00: 00.000	[{ "Цена": 21.99, "Количество": 3}, { "Цена": 22.99, "Количество": 2}, { "Цена": 23.99, "Количество": 2}]	22,99	2
SO43661	2011-06-01 00: 00: 00.000	[{ "Цена": 21.99, "Количество": 3}, { "Цена": 22.99, "Количество": 2}, { "Цена": 23.99, "Количество": 2}]	23,99	2

Прочитайте OPENJSON онлайн: https://riptutorial.com/ru/sql-server/topic/5030/openjson

глава 11: ParseName

Синтаксис

- PARSENAME ('object_name', object_piece)

параметры

'OBJECT_NAME'	object_piece
Имя объекта, для которого требуется получить указанную часть объекта. object_name - sysname. Этот параметр является необязательным именем объекта. Если все части имени объекта квалифицированы, это имя может содержать четыре части: имя сервера, имя базы данных, имя владельца и имя объекта.	Возвращается ли часть объекта. object_piece имеет тип int и может иметь следующие значения: 1 = имя объекта 2 = имя схемы 3 = имя базы данных 4 = имя сервера

Examples

ParseName

```
Declare @ObjectName nVarChar(1000)
Set @ObjectName = 'HeadOfficeSQL1.Northwind.dbo.Authors'

SELECT
 PARSENAME(@ObjectName, 4) as Server
,PARSENAME(@ObjectName, 3) as DB
,PARSENAME(@ObjectName, 2) as Owner
,PARSENAME(@ObjectName, 1) as Object
```

Возвращает:

сервер	база данных
HeadofficeSQL1	Северный ветер

владелец	объект
ПСЭ	Авторы

Прочитайте ParseName онлайн: https://riptutorial.com/ru/sql-server/topic/5775/parsename

глава 12: PHANTOM читать

Вступление

В системах баз данных изоляция определяет, как целостность транзакции видна другим пользователям и системам, поэтому она определяет, как / когда изменения, сделанные одной операцией, становятся видимыми для других. Фантомное чтение может возникать, когда вы получаете данные, еще не отправленные в базу данных.

замечания

Вы можете прочитать различные `ISOLATION LEVEL` на MSDN

Examples

Уровень изоляции READ UNCOMMITTED

Создать образец таблицы в примерной базе данных

```
CREATE TABLE [dbo].[Table_1](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [title] [varchar](50) NULL,
 CONSTRAINT [PK_Table_1] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS =
ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

Теперь откройте первый редактор запросов (в базе данных), вставьте приведенный ниже код и выполните (**не трогайте --rollback**), в этом случае вы вставляете строку в БД, но **не** совершаете изменений.

```
begin tran

INSERT INTO Table_1 values('Title 1')

SELECT * FROM [Test].[dbo].[Table_1]

--rollback
```

Теперь откройте второй редактор запросов (в базе данных), вставьте код ниже и выполните

```
begin tran
```

```
set transaction isolation level READ UNCOMMITTED

SELECT * FROM [Test].[dbo].[Table_1]
```

Вы можете заметить, что во втором редакторе вы можете увидеть только что созданную строку (но не зафиксированную) из первой транзакции. На первом редакторе выполните откат (выберите слово откат и выполните).

```
-- Rollback the first transaction
rollback
```

Выполните запрос во втором редакторе, и вы увидите, что запись исчезла (фантомное чтение), это происходит из-за того, что вы сообщаете второй транзакции, чтобы получить все строки, а также uncommitteds.

Это происходит, когда вы меняете уровень изоляции с помощью

```
set transaction isolation level READ UNCOMMITTED
```

Прочитайте PHANTOM читать онлайн: https://riptutorial.com/ru/sql-server/topic/8235/phantom-читать

глава 13: PIVOT / UNPIVOT

Синтаксис

- SELECT `<non-pivoted column>` ,
  [первый поворотный столбец] AS `<column name>` ,
  [второй поворотный столбец] AS `<column name>` ,
  ...
  [последний поворотный столбец] AS `<column name>`
  ОТ
  (`<SELECT query that produces the data>`)
  AS `<alias for the source query>`
  PIVOT
  (
  `<aggregation function>` (`<column being aggregated>`)
  ЗА
  [`<column that contains the values that will become column headers>`]
  IN ([первый поворотный столбец], [второй поворотный столбец],
  ... [последний поворотный столбец])
) AS `<alias for the pivot table>` `<optional ORDER BY clause>` ;

замечания

Используя операторы PIVOT и UNPIVOT, вы преобразовываете таблицу, перемещая строки (значения столбцов) таблицы в столбцы и наоборот. В рамках этого преобразования функции агрегации могут применяться к значениям таблицы.

Examples

Простая ось - статические столбцы

Используя таблицу продаж товаров из базы данных примеров , давайте вычислим и покажем общее количество, которое мы продали для каждого Продукта.

Это можно легко сделать с помощью группы, но давайте предположим, что мы «вращаем» нашу таблицу результатов так, чтобы для каждого идентификатора продукта был столбец.

```
SELECT [100], [145]
  FROM (SELECT ItemId , Quantity
          FROM #ItemSalesTable
       ) AS pivotIntermediate
 PIVOT (   SUM(Quantity)
           FOR ItemId IN ([100], [145])
       ) AS pivotTable
```

Поскольку наши «новые» столбцы - это числа (в исходной таблице), нам нужно заключить в квадратные скобки `[]`

Это даст нам

100	145
45	18

Простой PIVOT & UNPIVOT (T-SQL)

Ниже приведен простой пример, показывающий среднюю цену каждого товара за будний день.

Во-первых, предположим, что у нас есть таблица, которая хранит ежедневные отчеты о ценах всех предметов.

```
CREATE TABLE tbl_stock(item NVARCHAR(10), weekday NVARCHAR(10), price INT);

INSERT INTO tbl_stock VALUES
('Item1', 'Mon', 110), ('Item2', 'Mon', 230), ('Item3', 'Mon', 150),
('Item1', 'Tue', 115), ('Item2', 'Tue', 231), ('Item3', 'Tue', 162),
('Item1', 'Wed', 110), ('Item2', 'Wed', 240), ('Item3', 'Wed', 162),
('Item1', 'Thu', 109), ('Item2', 'Thu', 228), ('Item3', 'Thu', 145),
('Item1', 'Fri', 120), ('Item2', 'Fri', 210), ('Item3', 'Fri', 125),
('Item1', 'Mon', 122), ('Item2', 'Mon', 225), ('Item3', 'Mon', 140),
('Item1', 'Tue', 110), ('Item2', 'Tue', 235), ('Item3', 'Tue', 154),
('Item1', 'Wed', 125), ('Item2', 'Wed', 220), ('Item3', 'Wed', 142);
```

Таблица должна выглядеть следующим образом:

```
+========+=========+=======+
|   item | weekday | price |
+========+=========+=======+
|  Item1 |    Mon  |   110 |
+--------+---------+-------+
|  Item2 |    Mon  |   230 |
+--------+---------+-------+
|  Item3 |    Mon  |   150 |
+--------+---------+-------+
|  Item1 |    Tue  |   115 |
+--------+---------+-------+
|  Item2 |    Tue  |   231 |
+--------+---------+-------+
|  Item3 |    Tue  |   162 |
+--------+---------+-------+
|          . . .           |
+--------+---------+-------+
|  Item2 |    Wed  |   220 |
+--------+---------+-------+
|  Item3 |    Wed  |   142 |
+--------+---------+-------+
```

Чтобы выполнить агрегацию, которая должна найти среднюю цену за элемент за каждый недельный день, мы будем использовать реляционный оператор `PIVOT` чтобы повернуть столбец `weekday` табличного выражения в агрегированные значения строк, как `PIVOT` ниже:

```
SELECT * FROM tbl_stock
PIVOT (
    AVG(price) FOR weekday IN ([Mon], [Tue], [Wed], [Thu], [Fri])
) pvt;
```

Результат:

```
+--------+------+------+------+------+------+
|  item  |  Mon |  Tue |  Wed |  Thu |  Fri |
+--------+------+------+------+------+------+
|  Item1 |  116 |  112 |  117 |  109 |  120 |
|  Item2 |  227 |  233 |  230 |  228 |  210 |
|  Item3 |  145 |  158 |  152 |  145 |  125 |
+--------+------+------+------+------+------+
```

Наконец, чтобы выполнить обратную операцию `PIVOT` , мы можем использовать реляционный оператор `UNPIVOT` для поворота столбцов в строки, как `UNPIVOT` ниже:

```
SELECT * FROM tbl_stock
PIVOT (
    AVG(price) FOR weekday IN ([Mon], [Tue], [Wed], [Thu], [Fri])
) pvt
UNPIVOT (
    price FOR weekday IN ([Mon], [Tue], [Wed], [Thu], [Fri])
) unpvt;
```

Результат:

```
+=======+========+=========+
|  item |  price | weekday |
+=======+========+=========+
| Item1 |    116 |     Mon |
+-------+--------+---------+
| Item1 |    112 |     Tue |
+-------+--------+---------+
| Item1 |    117 |     Wed |
+-------+--------+---------+
| Item1 |    109 |     Thu |
+-------+--------+---------+
| Item1 |    120 |     Fri |
+-------+--------+---------+
| Item2 |    227 |     Mon |
+-------+--------+---------+
| Item2 |    233 |     Tue |
+-------+--------+---------+
| Item2 |    230 |     Wed |
+-------+--------+---------+
| Item2 |    228 |     Thu |
+-------+--------+---------+
| Item2 |    210 |     Fri |
+-------+--------+---------+
```

```
| Item3 |    145 |     Mon |
+-------+--------+---------+
| Item3 |    158 |     Tue |
+-------+--------+---------+
| Item3 |    152 |     Wed |
+-------+--------+---------+
| Item3 |    145 |     Thu |
+-------+--------+---------+
| Item3 |    125 |     Fri |
+-------+--------+---------+
```

Динамический PIVOT

Одна из проблем с запросом `PIVOT` заключается в том, что вам нужно указать все значения внутри выбора `IN` если вы хотите видеть их как столбцы. Быстрый способ обойти эту проблему - создать динамический выбор IN, `PIVOT` динамику `PIVOT` .

Для демонстрации мы будем использовать таблицу `Books` в базе данных `Bookstore` . Мы предполагаем, что таблица довольно де-нормирована и имеет следующие столбцы

```
Table: Books
----------------------------
BookId (Primary Key Column)
Name
Language
NumberOfPages
EditionNumber
YearOfPrint
YearBoughtIntoStore
ISBN
AuthorName
Price
NumberOfUnitsSold
```

Сценарий создания для таблицы будет выглядеть так:

```
CREATE TABLE [dbo].[BookList](
      [BookId] [int] NOT NULL,
      [Name] [nvarchar](100) NULL,
      [Language] [nvarchar](100) NULL,
      [NumberOfPages] [int] NULL,
      [EditionNumber] [nvarchar](10) NULL,
      [YearOfPrint] [int] NULL,
      [YearBoughtIntoStore] [int] NULL,
[NumberOfBooks] [int] NULL,
[ISBN] [nvarchar](30) NULL,
      [AuthorName] [nvarchar](200) NULL,
      [Price] [money] NULL,
      [NumberOfUnitsSold] [int] NULL,
 CONSTRAINT [PK_BookList] PRIMARY KEY CLUSTERED
(
      [BookId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS =
ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
GO
```

Теперь, если нам нужно запросить базу данных и выяснить количество книг на английском, русском, немецком, хинди, латинском языках, которые каждый год покупаются в книжном магазине и представляют нашу продукцию в небольшом формате отчета, мы можем использовать запрос PIVOT, подобный этому

```
SELECT * FROM
  (
   SELECT YearBoughtIntoStore AS [Year Bought],[Language], NumberOfBooks
   FROM BookList
  ) sourceData
 PIVOT
  (
  SUM(NumberOfBooks)
  FOR [Language] IN (English, Russian, German, Hindi, Latin)
  ) pivotrReport
```

Особый случай - когда у нас нет полного списка языков, поэтому мы будем использовать динамический SQL, как показано ниже.

```
DECLARE @query VARCHAR(4000)
DECLARE @languages VARCHAR(2000)
SELECT @languages =
        STUFF((SELECT DISTINCT '],['+LTRIM([Language])FROM [dbo].[BookList]
        ORDER BY '],['+LTRIM([Language]) FOR XML PATH('') ),1,2,'') + ']'
SET @query=
'SELECT * FROM
  (SELECT YearBoughtIntoStore AS [Year Bought],[Language],NumberOfBooks
   FROM BookList) sourceData
PIVOT(SUM(NumberOfBooks)FOR [Language] IN ('+ @languages +')) pivotrReport' EXECUTE(@query)
```

Прочитайте PIVOT / UNPIVOT онлайн: https://riptutorial.com/ru/sql-server/topic/591/pivot---unpivot

глава 14: Schemas

Examples

Создание схемы

```
CREATE SCHEMA dvr AUTHORIZATION Owner
    CREATE TABLE sat_Sales (source int, cost int, partid int)
    GRANT SELECT ON SCHEMA :: dvr TO  User1
    DENY SELECT ON SCHEMA :: dvr to User 2
GO
```

Изменить схему

```
ALTER SCHEMA dvr
    TRANSFER dbo.tbl_Staging;
GO
```

Это перенесет таблицу tbl_Staging из схемы dbo в схему dvr

Удаление схем

```
DROP SCHEMA dvr
```

Цель

Схема относится к конкретным таблицам базы данных и тому, как они связаны друг с другом. Он обеспечивает организационную схему построения базы данных. Дополнительные преимущества реализации схем баз данных заключаются в том, что схемы могут использоваться как метод, ограничивающий / предоставляющий доступ к определенным таблицам в базе данных.

Прочитайте Schemas онлайн: https://riptutorial.com/ru/sql-server/topic/5806/schemas

глава 15: SCOPE_IDENTITY ()

Синтаксис

- SELECT SCOPE_IDENTITY ();
- SELECT SCOPE_IDENTITY () AS [SCOPE_IDENTITY];
- SCOPE_IDENTITY ()

Examples

Введение с простым примером

SCOPE_IDENTITY () возвращает последнее значение идентификации, вставленное в столбец идентификатора в той же области. Объем - это модуль: хранимая процедура, триггер, функция или партия. Поэтому два оператора находятся в одной области, если они находятся в одной и той же хранимой процедуре, функции или партии.

INSERT INTO ([column1], [column2]) VALUES (8,9);
ИДТИ
SELECT SCOPE_IDENTITY () AS [SCOPE_IDENTITY];
ИДТИ

Прочитайте SCOPE_IDENTITY () онлайн: https://riptutorial.com/ru/sql-server/topic/5326/scope-identity---

глава 16: SQL Server Evolution через различные версии (2000 - 2016)

Вступление

Я использую SQL Server с 2004 года. Я начал работать с 2000 года, и теперь я собираюсь использовать SQL Server 2016. Я создал таблицы, представления, функции, триггеры, хранимые процедуры и написал много SQL-запросов, но я не использовал много новых функций из последующих версии. Я googled это, но, к сожалению, я не нашел все функции в одном месте. Поэтому я собрал и подтвердил эту информацию из разных источников и разместил здесь. Я просто добавляю информацию высокого уровня для всех версий, начиная с 2000 до 20

Examples

SQL Server версии 2000 - 2016

Следующие функции добавлены в SQL Server 2000 из предыдущей версии:

1. Добавлены новые типы данных (BIGINT, SQL_VARIANT, TABLE)
2. Вместо триггеров были введены как продвижение к DDL.
3. Каскадная ссылочная целостность.
4. Поддержка XML
5. Пользовательские функции и виды разделов.
6. Индексированные представления (разрешение индекса на представления с вычисленными столбцами).

Следующие функции добавлены в версию 2005 из предыдущей версии:

1. Улучшение в TOP-разделе с опцией «WITH TIES».
2. Команды манипуляции с данными (DML) и OUTPUT, чтобы получить значения INSERTED и DELETED
3. Операторы PIVOT и UNPIVOT.
4. Обработка исключений с блоком TRY / CATCH
5. Функции ранжирования
6. Общие выражения таблицы (CTE)
7. Common Language Runtime (интеграция языков .NET для создания таких объектов, как хранимые процедуры, триггеры, функции и т. Д.)
8. Сервисный брокер (обработка сообщения между отправителем и получателем в слабосвязанной манере)
9. Шифрование данных (собственные возможности для поддержки шифрования данных,

хранящихся в пользовательских базах данных)
10. SMTP-почта
11. Конечные точки HTTP (создание конечных точек с использованием простого оператора T-SQL, отображающего объект, доступ к которому осуществляется через Интернет)
12. Несколько наборов активных результатов (MARS). Это позволяет постоянному подключению базы данных от одного клиента иметь более одного активного запроса на соединение.
13. Службы интеграции SQL Server (будут использоваться в качестве основного инструмента ETL (извлечение, преобразование и загрузка)
14. Улучшения в службах Analysis Services и Reporting Services.
15. Разбиение таблиц и индексов. Позволяет разбивать таблицы и индексы на основе границ раздела, как определено ФУНКЦИЕЙ PARTITION, с отдельными разделами, сопоставленными файловыми группами через схему PARTITION SCHEME.

Следующие функции добавлены в версию 2008 из предыдущей версии:

1. Улучшение существующих типов данных DATE и TIME
2. Новые функции, такие как SYSUTCDATETIME () и SYSDATETIMEOFFSET ()
3. Запасные столбцы - для экономии значительного объема дискового пространства.
4. Большие пользовательские типы (до 2 ГБ)
5. Введена новая функция для передачи типа данных таблицы в хранимые процедуры и функции
6. Новая команда MERGE для операций INSERT, UPDATE и DELETE
7. Новый тип данных HierarchyID
8. Пространственные типы данных - для представления физического местоположения и формы любого геометрического объекта.
9. Более быстрые запросы и отчеты с помощью GROUPING SETS - расширение для предложения GROUP BY.
10. Усовершенствование опции хранения FILESTREAM

Следующие функции добавлены в версию 2008 R2 из предыдущей версии:

1. PowerPivot - для обработки больших наборов данных.
2. Построитель отчетов 3.0
3. Облако готово
4. StreamInsight
5. Основные данные
6. Интеграция с SharePoint
7. DACPAC (пакеты компонентов приложений уровня данных)
8. Улучшение других функций SQL Server 2008

Следующие функции добавлены в версию 2012 из предыдущей версии:

1. Индексы хранилища столбцов - уменьшают использование ввода-вывода и памяти в

больших запросах.
2. Pagination - разбиение на страницы можно сделать с помощью команд «OFFSET» и « FETCH».
3. Содержащаяся база данных - отличная функция для периодической миграции данных.
4. Группы доступности AlwaysOn
5. Поддержка ядра Windows Server
6. Определенные пользователем роли сервера
7. Поддержка больших данных
8. PowerView
9. Улучшения SQL Azure
10. Табличная модель (SSAS)
11. Услуги по обеспечению качества данных DQS
12. Таблица файлов - усовершенствование функции FILESTREAM, которая была представлена в 2008 году.
13. Улучшение обработки ошибок, включая инструкцию THROW
14. Улучшение SQL Server Management Studio Отладка a. SQL Server 2012 вводит больше опций для контроля контрольных точек. б. Улучшения окон в режиме отладки с. Усовершенствование в IntelliSense - например, вставка фрагментов кода.

Следующие функции добавлены в версию 2014 из предыдущей версии:

1. Встроенный в память модуль OLTP - повышает производительность до 20 раз.
2. Улучшения AlwaysOn
3. Расширение буферного пула
4. Гибридные облачные функции
5. Улучшение индексов хранилища столбцов (например, индексы хранилища индексируемых индексов)
6. Улучшения обработки запросов (например, SELECT INTO)
7. Power BI для интеграции Office 365
8. Отсроченная долговечность
9. Усовершенствования для резервного копирования баз данных

Следующие функции добавлены в версию 2016 из предыдущей версии:

1. Always Encrypted - Always Encrypted предназначен для защиты данных в состоянии покоя или в движении.
2. Оперативная аналитика в режиме реального времени
3. PolyBase в SQL Server
4. Поддержка родного JSON
5. Магазин запросов
6. Усовершенствования AlwaysOn
7. Улучшенный OLTP-файл с памятью
8. Несколько файлов базы данных TempDB

9. Растяжка базы данных
10. Безопасность уровня строки
11. Улучшения в памяти

    Расширения T-SQL или новые дополнения в SQL Server 2016

1. TRUNCATE TABLE с PARTITION
2. DROP IF EXISTS
3. Функции STRING_SPLIT и STRING_ESCAPE
4. ALTER TABLE теперь может изменять многие столбцы, пока таблица остается в сети, используя WITH (ONLINE = ON | OFF).
5. MAXDOP для DBCC CHECKDB, DBCC CHECKTABLE и DBCC CHECKFILEGROUP
6. ALTER DATABASE SET AUTOGROW_SINGLE_FILE
7. ALTER DATABASE SET AUTOGROW_ALL_FILES
8. Функции COMPRESS и DECOMPRESS
9. ЗАЯВЛЕНИЕ FORMATMESSAGE
10. 2016 вводит еще 8 объектов с SERVERPROPERTY

а. InstanceDefaultDataPath

б. InstanceDefaultLogPath

с. ProductBuild

д. ProductBuildType

е. ProductMajorVersion

е. ProductMinorVersion

г. ProductUpdateLevel

час ProductUpdateReference

Прочитайте SQL Server Evolution через различные версии (2000 - 2016) онлайн: https://riptutorial.com/ru/sql-server/topic/10129/sql-server-evolution-через-различные-версии--2000---2016-

глава 17: SQLCMD

замечания

Вам необходимо либо находиться на пути, где существует SQLCMD.exe, либо добавить его в переменную среды PATH.

Examples

SQLCMD.exe вызывается из командного файла или командной строки

```
echo off

cls

sqlcmd.exe -S "your server name" -U "sql user name" -P "sql password" -d "name of databse" -Q
"here you may write your query/stored procedure"
```

Пакетные файлы, подобные этим, могут использоваться для автоматизации задач, например, для создания резервных копий баз данных в указанное время (может быть запланировано с помощью планировщика заданий) для версии SQL Server Express, в которой нельзя использовать приложения агента.

Прочитайте SQLCMD онлайн: https://riptutorial.com/ru/sql-server/topic/5396/sqlcmd

глава 18: UNION

Examples

Союз и союз все

Операция **Union** объединяет результаты двух или нескольких запросов в один результирующий набор , который включает в себя все строки , которые принадлежат ко всем запросам в объединении и будет игнорировать все дубликаты , которые существуют. **Союз все** также делает то же самое, но включает даже повторяющиеся значения. Концепция операции объединения будет очевидна из приведенного ниже примера. Мало что нужно учитывать при использовании союза:

1. Количество и порядок столбцов должны быть одинаковыми во всех запросах.

2. Типы данных должны быть совместимы.

Пример:

У нас есть три таблицы: Marksheet1, Marksheet2 и Marksheet3. Marksheet3 является дублирующей таблицей Marksheet2, которая содержит те же значения, что и Marksheet2.

Таблица 1 : Таблица 1

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

Таблица 2 : **Таблица** 2

CourseCode	CourseName	MarksObtained
201	PhysicsII	82
202	ChemistryII	86
203	MathsII	95
204	EnglishII	70
205	ComputerII	86

Таблица 3 : Табличка 3

SubjectCode	SubjectName	MarksObtained
201	PhysicsII	82
202	ChemistryII	86
203	MathsII	95
204	EnglishII	70
205	ComputerII	86

Союз на таблицах Marksheet1 и Marksheet2

```
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet1
UNION
SELECT CourseCode, CourseName, MarksObtained
FROM Marksheet2
```

Примечание. Результат объединения трех таблиц также будет таким же, как объединение в Marksheet1 и Marksheet2, потому что операция объединения не принимает повторяющихся значений.

```
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet1
UNION
SELECT CourseCode, CourseName, MarksObtained
FROM Marksheet2
UNION
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet3
```

ВЫХОД

	SubjectCode	SubjectName	MarksObtained
1	101	Physics	87
2	102	Chemistry	75
3	103	Maths	85
4	104	English	89
5	105	Computer	95
6	201	PhysicsII	82
7	202	ChemistryII	86
8	203	MathsII	95
9	204	EnglishII	70
10	205	ComputerII	86

Союз Все

```
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet1
UNION ALL
SELECT CourseCode, CourseName, MarksObtained
FROM Marksheet2
UNION ALL
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet3
```

ВЫХОД

	SubjectCode	SubjectName	MarksObtained
1	101	Physics	87
2	102	Chemistry	75
3	103	Maths	85
4	104	English	89
5	105	Computer	95
6	201	PhysicsII	82
7	202	ChemistryII	86
8	203	MathsII	95
9	204	EnglishII	70
10	205	ComputerII	86
11	201	PhysicsII	82
12	202	ChemistryII	86
13	203	MathsII	95
14	204	EnglishII	70
15	205	ComputerII	86

Здесь вы заметите, что повторяющиеся значения из Marksheet3 также отображаются с использованием union all.

Прочитайте UNION онлайн: https://riptutorial.com/ru/sql-server/topic/5590/union

глава 19: Анализ запроса

Examples

Scan vs Seek

При просмотре плана выполнения вы можете увидеть, что SQL Server решил выполнить поиск или сканирование.

Иск возникает, когда SQL Server знает, куда ему нужно идти, и захватывать только определенные элементы. Обычно это происходит, когда хорошие фильтры помещаются в запрос, например, `where name = 'Foo'` .

Сканирование происходит, когда SQL Server не знает точно, где именно нужны все данные, или решил, что сканирование будет более эффективным, чем поиск, если будет выбрано достаточно данных.

Иски, как правило, быстрее, поскольку они только захватывают подраздел данных, тогда как сканы выбирают большинство данных.

Прочитайте Анализ запроса онлайн: https://riptutorial.com/ru/sql-server/topic/7713/анализ-запроса

глава 20: Безопасность на уровне строк

Examples

Предикат фильтра RLS

База данных Sql Server 2016+ и Azure Sql позволяет автоматически фильтровать строки, которые возвращаются в инструкции select с использованием некоторого предиката. Эта функция называется **безопасностью уровня Row** .

Во-первых, вам нужна табличная функция, содержащая некоторый предикат, который описывает, что это условие, которое позволит пользователям читать данные из какой-либо таблицы:

```
DROP FUNCTION IF EXISTS dbo.pUserCanAccessCompany
GO
CREATE FUNCTION

dbo.pUserCanAccessCompany(@CompanyID int)

    RETURNS TABLE
    WITH SCHEMABINDING
AS RETURN (
    SELECT 1 as canAccess WHERE

    CAST(SESSION_CONTEXT(N'CompanyID') as int) = @CompanyID

)
```

В этом примере предикат говорит, что доступ к компании могут получить только пользователи, имеющие значение в SESSION_CONTEXT, соответствующее совпадающему входному аргументу. Вы можете поместить любое другое условие, например, проверять роль базы данных или базу данных текущего пользователя и т. Д.

> Большая часть приведенного выше кода - это шаблон, который вы скопируете. Единственное, что здесь изменится, это имя и аргументы предиката и условия в предложении WHERE. Теперь вы создаете политику безопасности, которая будет применять этот предикат в некоторой таблице.

Теперь вы можете создать политику безопасности, которая будет применять предикат в некоторой таблице:

```
CREATE SECURITY POLICY dbo.CompanyAccessPolicy
    ADD FILTER PREDICATE dbo.pUserCanAccessCompany(CompanyID) ON dbo.Company
    WITH (State=ON)
```

Эта политика безопасности присваивает предикат таблице компании. Всякий раз, когда

кто-то пытается прочитать данные из таблицы компании, политика безопасности будет применять предикат для каждой строки, передать столбец CompanyID в качестве параметра предиката, а предикат будет оценивать, должна ли эта строка быть возвращена в результате запроса SELECT.

Изменение политики безопасности RLS

Политика безопасности - это группа предикатов, связанных с таблицами, которые могут управляться вместе. Вы можете добавлять или удалять предикаты или включать / выключать всю политику.

Вы можете добавить больше предикатов в таблицы в существующую политику безопасности.

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy
    ADD FILTER PREDICATE dbo.pUserCanAccessCompany(CompanyID) ON dbo.Company
```

Вы можете удалить некоторые предикаты из политики безопасности:

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy
    DROP FILTER PREDICATE ON dbo.Company
```

Вы можете отключить политику безопасности

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy WITH ( STATE = OFF );
```

Вы можете включить политику безопасности, которая была отключена:

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy WITH ( STATE = ON );
```

Предотвращение обновления с использованием предиката блока RLS

Безопасность на уровне строк позволяет определить некоторые предикаты, которые будут контролировать, кто может обновлять строки в таблице. Сначала вам нужно определить некоторую функцию table-value, которая представляет предикат, который будет контролировать политику доступа wll.

СОЗДАТЬ ФУНКЦИЮ

dbo.pUserCanAccessProduct (@CompanyID int)

```
RETURNS TABLE
WITH SCHEMABINDING
```

AS RETURN (SELECT 1 as canAccess WHERE

CAST (SESSION_CONTEXT (N'CompanyID)) как int) = @CompanyID

) В этом примере предикат говорит, что доступ к компании могут получить только пользователи, имеющие значение в SESSION_CONTEXT, соответствующее совпадающему входному аргументу. Вы можете поместить любое другое условие, например, проверять роль базы данных или базу данных текущего пользователя и т. Д.

> Большая часть приведенного выше кода - это шаблон, который вы скопируете. Единственное, что здесь изменится, это имя и аргументы предиката и условия в предложении WHERE. Теперь вы создаете политику безопасности, которая будет применять этот предикат в некоторой таблице.

Теперь мы можем создать политику безопасности с предикатом, который будет блокировать обновления в таблице продуктов, если столбец CompanyID в таблице не удовлетворяет предикату.

СОЗДАТЬ ПОЛИТИКУ БЕЗОПАСНОСТИ dbo.ProductAccessPolicy ADD BLOCK PREDICATE dbo.pUserCanAccessProduct (CompanyID) ON dbo.Product

Этот предикат будет применяться ко всем операциям. Если вы хотите применить предикат к некоторой операции, вы можете написать что-то вроде:

СОЗДАТЬ ПОЛИТИКУ БЕЗОПАСНОСТИ dbo.ProductAccessPolicy ADD BLOCK PREDICATE dbo.pUserCanAccessProduct (CompanyID) ON dbo.Product ПОСЛЕ ВСТАВКИ

Возможные параметры, которые вы можете добавить после определения предиката блока:

[{ПОСЛЕ {ВСТАВЬТЕ | ОБНОВИТЬ } }
| {ПЕРЕД ДОПОЛНЕНИЕМ | УДАЛЯТЬ } }]

Прочитайте Безопасность на уровне строк онлайн: https://riptutorial.com/ru/sql-server/topic/7045/безопасность-на-уровне-строк

глава 21: Временные таблицы

замечания

SQL Server 2016 вводит поддержку временных версий системной версии как функцию базы данных, которая обеспечивает встроенную поддержку для предоставления информации о данных, хранящихся в таблице, в любой момент времени, а не только данных, которые являются правильными в текущий момент времени.

Временная таблица с версией в системе - это новый тип пользовательской таблицы в SQL Server 2016, предназначенный для сохранения полной истории изменений данных и облегчения анализа во времени. Этот тип временной таблицы называется временной версией с системной версией, так как период действия для каждой строки управляется системой (т.е. движком базы данных). Каждая временная таблица имеет два явно определенных столбца, каждый из которых имеет тип данных datetime2. Эти столбцы называются столбцами периода. Эти столбцы периода используются исключительно системой для записи периода действия для каждой строки всякий раз, когда изменяется строка.

Examples

СОЗДАТЬ временные таблицы

```
CREATE TABLE dbo.Employee
(
  [EmployeeID] int NOT NULL PRIMARY KEY CLUSTERED
  , [Name] nvarchar(100) NOT NULL
  , [Position] varchar(100) NOT NULL
  , [Department] varchar(100) NOT NULL
  , [Address] nvarchar(1024) NOT NULL
  , [AnnualSalary] decimal (10,2) NOT NULL
  , [ValidFrom] datetime2 (2) GENERATED ALWAYS AS ROW START
  , [ValidTo] datetime2 (2) GENERATED ALWAYS AS ROW END
  , PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo)
 )
 WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.EmployeeHistory));
```

INSERTS: В **INSERT** система устанавливает значение для столбца **ValidFrom** в момент начала текущей транзакции (в часовом поясе UTC) на основе системного тактового сигнала и присваивает значение для столбца **ValidTo** максимальному значению 9999- 12-31. Это знаменует строку открытой.

ОБНОВЛЕНИЯ: В **UPDATE** система сохраняет предыдущее значение строки в таблице истории и устанавливает значение для столбца **ValidTo** в момент начала текущей транзакции (в часовом поясе UTC) на основе системных часов. Это означает, что строка

закрыта, с периодом, для которого строка была действительной. В текущей таблице строка обновляется с ее новым значением, и система устанавливает значение для столбца **ValidFrom** для начала транзакции (в часовом поясе UTC) на основе системных часов. Значение обновленной строки в текущей таблице для столбца **ValidTo** остается максимальным значением 9999-12-31.

DELETES : В **DELETE** система сохраняет предыдущее значение строки в таблице предыстории и устанавливает значение для столбца **ValidTo** в момент начала текущей транзакции (в часовом поясе UTC) на основе системных часов. Это означает, что строка закрыта, с периодом, для которого была выполнена предыдущая строка. В текущей таблице строка удаляется. Запросы текущей таблицы не возвратят эту строку. Только запросы, которые обрабатывают данные истории, возвращают данные, для которых строка закрыта.

MERGE : В **MERGE** операция ведет себя точно так, как если бы выполнялось до трех операторов (**INSERT** , **UPDATE** и / или **DELETE**), в зависимости от того, что указано как действия в операторе **MERGE** .

Совет. Время, записанное в столбцах системы datetime2, основано на времени начала самой транзакции. Например, все строки, вставленные в одну транзакцию, будут иметь одинаковое время UTC, записанное в столбце, соответствующее началу периода **SYSTEM_TIME** .

Как мне запрашивать временные данные?

```
SELECT * FROM Employee
    FOR SYSTEM_TIME
        BETWEEN '2014-01-01 00:00:00.0000000' AND '2015-01-01 00:00:00.0000000'
            WHERE EmployeeID = 1000 ORDER BY ValidFrom;
```

Вернуть фактическое значение, указанное в момент времени (FOR SYSTEM_TIME AS OF)

Возвращает таблицу со строками, содержащими значения, которые были действительными (текущими) в указанный момент времени в прошлом.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME AS  OF '2016-08-06 08:32:37.91'
```

ДЛЯ СИСТЕМЫ_TIME МЕЖДУ А ТАКЖЕ

То же, что указано выше в описании FOR SYSTEM_TIME FROM <start_date_time> TO <end_date_time>, за исключением того, что таблица возвращаемых строк включает строки, которые стали активными на верхней границе, определенной конечной точкой < end_date_time>.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME BETWEEN  '2015-01-01' AND '2015-12-31'
```

FOR SYSTEM_TIME FROM K

Возвращает таблицу со значениями для всех версий строк, которые были активны в пределах указанного временного диапазона, независимо от того, начали ли они быть активными до значения параметра <start_date_time> для аргумента FROM или перестали быть активными после значения параметра <end_date_time> для K аргументу. Внутри соединение выполняется между временной таблицей и ее исторической таблицей, и результаты фильтруются, чтобы возвращать значения для всех версий строк, которые были активны в любое время в течение указанного диапазона времени. Строки, которые активировались точно на нижней границе, определяемой конечной точкой FROM, включены и записи, которые стали активными точно на верхней границе, определяемой конечной точкой TO, не включены.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME FROM '2015-01-01' TO '2015-12-31'
```

ДЛЯ SYSTEM_TIME СОДЕРЖАЩИХСЯ (,)

Возвращает таблицу со значениями для всех версий строк, которые были открыты и закрыты в течение заданного интервала времени, определяемого двумя значениями datetime для аргумента CONTAINED IN. Строки, которые активировались точно на нижней границе или перестали быть активными точно на верхней границе, включены.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME CONTAINED IN ('2015-04-01', '2015-09-25')
```

ДЛЯ SYSTEM_TIME ALL

Возвращает объединение строк, относящихся к текущей и таблице истории.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME ALL
```

Создание временной таблицы с параметрами, оптимизированной по параметрам памяти, и очистка таблицы истории SQL Server

Создание временной таблицы с таблицей истории по умолчанию является удобным вариантом, когда вы хотите управлять именованием и по-прежнему полагаться на систему для создания таблицы истории с настройкой по умолчанию. В приведенном ниже примере представлена новая временная таблица с оптимизированной памятью, связанная с версией, связанная с новой таблицей истории диска.

```
CREATE SCHEMA History
GO
CREATE TABLE dbo.Department
(
    DepartmentNumber char(10) NOT NULL PRIMARY KEY NONCLUSTERED,
    DepartmentName varchar(50) NOT NULL,
    ManagerID int  NULL,
    ParentDepartmentNumber char(10) NULL,
    SysStartTime datetime2 GENERATED ALWAYS AS ROW START HIDDEN NOT NULL,
    SysEndTime datetime2 GENERATED ALWAYS AS ROW END HIDDEN NOT NULL,
    PERIOD FOR SYSTEM_TIME (SysStartTime,SysEndTime)
)
WITH
    (
        MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA,
        SYSTEM_VERSIONING = ON ( HISTORY_TABLE = History.DepartmentHistory )
    );
```

Очистка таблицы истории SQL Server Со временем таблица истории может значительно увеличиться. Поскольку вставка, обновление или удаление данных из таблицы истории не разрешена, единственный способ очистить таблицу истории - сначала отключить системное управление версиями:

```
ALTER TABLE dbo.Employee
```

SET (SYSTEM_VERSIONING = OFF); ИДТИ

Удалите ненужные данные из таблицы истории:

```
    DELETE FROM dbo.EmployeeHistory
```

WHERE EndTime <= '2017-01-26 14:00:29';

а затем снова включить системное управление версиями:

```
ALTER TABLE dbo.Employee
```

SET (SYSTEM_VERSIONING = ON (HISTORY_TABLE = [dbo]. [EmployeeHistory], DATA_CONSISTENCY_CHECK = ON));

Очистка таблицы истории в Базах данных Azure SQL немного отличается, поскольку базы данных Azure SQL имеют встроенную поддержку для очистки таблицы истории. Во-первых, учетная запись временной истории должна быть включена на уровне базы данных:

```
ALTER DATABASE CURRENT
```

SET TEMPORAL_HISTORY_RETENTION ON GO

Затем установите период хранения для каждой таблицы:

```
ALTER TABLE dbo.Employee
```

SET (SYSTEM_VERSIONING = ON (HISTORY_RETENTION_PERIOD = 90 DAYS));

Это приведет к удалению всех данных в таблице предыстории старше 90 дней. Внутренние базы данных SQL Server 2016 не поддерживают TEMPORAL_HISTORY_RETENTION и HISTORY_RETENTION_PERIOD, и любой из этих двух запросов выполняется на локальных серверах SQL Server 2016, при этом будут возникать следующие ошибки.

Ошибка TEMPORAL_HISTORY_RETENTION:

```
Msg 102, Level 15, State 6, Line 34
```

Неверный синтаксис рядом с 'TEMPORAL_HISTORY_RETENTION'.

Для ошибки HISTORY_RETENTION_PERIOD будет:

```
Msg 102, Level 15, State 1, Line 39
```

Неверный синтаксис рядом с 'HISTORY_RETENTION_PERIOD'.

Прочитайте Временные таблицы онлайн: https://riptutorial.com/ru/sql-server/topic/5296/временные-таблицы

глава 22: Вставить

Examples

Добавить строку в таблицу с именем Invoices

```
INSERT INTO Invoices [ /* column names may go here */ ]
VALUES (123, '1234abc', '2016-08-05 20:18:25.770', 321, 5, '2016-08-04');
```

- Имена столбцов требуются, если таблица, в которую вы вставляете, содержит столбец с атрибутом IDENTITY.

```
INSERT INTO Invoices ([ID], [Num], [DateTime], [Total], [Term], [DueDate])
VALUES (123, '1234abc', '2016-08-05 20:18:25.770', 321, 5, '2016-08-25');
```

Прочитайте Вставить онлайн: https://riptutorial.com/ru/sql-server/topic/5323/вставить

глава 23: ВСТАВИТЬ В

Вступление

Инструкция INSERT INTO используется для вставки новых записей в таблицу.

Examples

Таблица INSERT Hello World INTO

```
CREATE TABLE MyTableName
(
    Id INT,
    MyColumnName NVARCHAR(1000)
)
GO

INSERT INTO MyTableName (Id, MyColumnName)
VALUES (1, N'Hello World!')
GO
```

ВСТАВИТЬ на определенные столбцы

Чтобы сделать вставку для определенных столбцов (в отличие от всех), вы должны указать столбцы, которые хотите обновить.

```
INSERT INTO USERS (FIRST_NAME, LAST_NAME)
VALUES ('Stephen', 'Jiang');
```

Это будет работать только в том случае, если столбцы, которые вы не указали, имеют значение NULL, идентификатор, тип данных timestamp или вычисленные столбцы; или столбцы, которые имеют ограничение по умолчанию. Поэтому, если какой-либо из них не является нулевым, неидентифицированным, не timestamp, не вычисленным, не по умолчанию столбцами с оценкой ... затем попытка такого типа вставки вызовет сообщение об ошибке, сообщающее вам, что вы должны предоставить значение для применимых полей.

Вставить несколько строк данных

Вставка нескольких строк данных в SQL Server 2008 или более поздних версиях:

```
INSERT INTO USERS VALUES
(2, 'Michael', 'Blythe'),
(3, 'Linda', 'Mitchell'),
(4, 'Jillian', 'Carson'),
(5, 'Garrett', 'Vargas');
```

Чтобы вставить несколько строк данных в более ранние версии SQL Server, используйте « UNION ALL» следующим образом:

```
INSERT INTO USERS (FIRST_NAME, LAST_NAME)
SELECT 'James', 'Bond' UNION ALL
SELECT 'Miss', 'Moneypenny' UNION ALL
SELECT 'Raoul', 'Silva'
```

Обратите внимание, что ключевое слово «INTO» является необязательным в запросах INSERT. Еще одно предупреждение состоит в том, что SQL-сервер поддерживает только 1000 строк в одном INSERT, поэтому вам приходится разбивать их партиями.

ВСТАВИТЬ одну строку данных

Одна строка данных может быть вставлена двумя способами:

```
INSERT INTO USERS(Id, FirstName, LastName)
VALUES (1, 'Mike', 'Jones');
```

Или же

```
INSERT INTO USERS
VALUES (1, 'Mike', 'Jones');
```

Обратите внимание, что второй оператор insert только позволяет значениям точно в том же порядке, что и столбцы таблицы, тогда как в первой вставке порядок значений можно изменить следующим образом:

```
INSERT INTO USERS(FirstName, LastName, Id)
VALUES ('Mike', 'Jones', 1);
```

Используйте OUTPUT для получения нового идентификатора

Когда INSERTING вы можете использовать `OUTPUT INSERTED.ColumnName` для получения значений из недавно вставленной строки, например, вновь созданного Id - полезно, если у вас есть столбец `IDENTITY` или любое значение по умолчанию или вычисленное значение.

При программном вызове этого (например, с ADO.net) вы рассматриваете его как обычный запрос и читаете значения так, как если бы вы сделали `SELECT` состояние.

```
-- CREATE TABLE OutputTest ([Id] INT NOT NULL PRIMARY KEY IDENTITY, [Name] NVARCHAR(50))

INSERT INTO OutputTest ([Name])
OUTPUT INSERTED.[Id]
VALUES ('Testing')
```

Если идентификатор недавно добавленной строки требуется внутри одного и того же

набора запросов или хранимой процедуры.

```
-- CREATE a table variable having column with the same datatype of the ID

DECLARE @LastId TABLE ( id int);

INSERT INTO OutputTest ([Name])
OUTPUT INSERTED.[Id] INTO @LastId
VALUES ('Testing')

SELECT id FROM @LastId

-- We can set the value in a variable and use later in procedure

DECLARE @LatestId int = (SELECT id FROM @LastId)
```

INSERT из результатов запроса SELECT

Чтобы вставить данные, полученные из SQL-запроса (одно или несколько строк)

```
INSERT INTO Table_name (FirstName, LastName, Position)
SELECT FirstName, LastName, 'student' FROM Another_table_name
```

Примечание. «Student» в SELECT - это строковая константа, которая будет вставлена в каждую строку.

При необходимости вы можете выбрать и вставить данные из / в ту же таблицу

Прочитайте ВСТАВИТЬ В онлайн: https://riptutorial.com/ru/sql-server/topic/3814/вставить-в

глава 24: Встроенная память OLTP (Hekaton)

Examples

Создание оптимизированной таблицы памяти

```
-- Create demo database
CREATE DATABASE SQL2016_Demo
 ON  PRIMARY
(
    NAME = N'SQL2016_Demo',
    FILENAME = N'C:\Dump\SQL2016_Demo.mdf',
    SIZE = 5120KB,
    FILEGROWTH = 1024KB
 )
 LOG ON
 (
    NAME = N'SQL2016_Demo_log',
    FILENAME = N'C:\Dump\SQL2016_Demo_log.ldf',
    SIZE = 1024KB,
    FILEGROWTH = 10%
 )
GO

use SQL2016_Demo
go

-- Add Filegroup by MEMORY_OPTIMIZED_DATA type
ALTER DATABASE SQL2016_Demo
    ADD FILEGROUP MemFG CONTAINS MEMORY_OPTIMIZED_DATA
GO


--Add a file to defined filegroup
ALTER DATABASE SQL2016_Demo ADD FILE
    (
        NAME = MemFG_File1,
        FILENAME = N'C:\Dump\MemFG_File1' -- your file path, check directory exist before
executing this code
    )
TO FILEGROUP MemFG
GO

--Object Explorer -- check database created
GO

-- create memory optimized table 1
CREATE TABLE dbo.MemOptTable1
(
    Column1     INT         NOT NULL,
    Column2     NVARCHAR(4000)  NULL,
    SpidFilter  SMALLINT    NOT NULL   DEFAULT (@@spid),

    INDEX ix_SpidFiler NONCLUSTERED (SpidFilter),
```

```
    INDEX ix_SpidFilter HASH (SpidFilter) WITH (BUCKET_COUNT = 64),

    CONSTRAINT CHK_soSessionC_SpidFilter
        CHECK ( SpidFilter = @@spid ),
)
    WITH
        (MEMORY_OPTIMIZED = ON,
         DURABILITY = SCHEMA_AND_DATA);  --or DURABILITY = SCHEMA_ONLY
go

-- create memory optimized table 2
CREATE TABLE MemOptTable2
(
    ID INT NOT NULL PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET_COUNT = 10000),
    FullName NVARCHAR(200) NOT NULL,
    DateAdded DATETIME NOT NULL
) WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA)
GO
```

Показать созданные DLL-файлы и таблицы для оптимизированных по памяти таблиц

```
SELECT
    OBJECT_ID('MemOptTable1') AS MemOptTable1_ObjectID,
    OBJECT_ID('MemOptTable2') AS MemOptTable2_ObjectID
GO

SELECT
    name,description
FROM sys.dm_os_loaded_modules
WHERE name LIKE '%XTP%'
GO
```

Показать все таблицы с оптимизацией памяти:

```
SELECT
    name,type_desc,durability_desc,Is_memory_Optimized
FROM sys.tables
    WHERE Is_memory_Optimized = 1
GO
```

Оптимизированные по таблице типы таблиц и таблицы Temp

Например, это традиционный тип таблицы на основе tempdb:

```
CREATE TYPE dbo.testTableType AS TABLE
(
   col1 INT NOT NULL,
   col2 CHAR(10)
);
```

Чтобы оптимизировать этот тип таблицы, просто добавьте параметр `memory_optimized=on` и добавьте индекс, если в исходном типе нет:

```
CREATE TYPE dbo.testTableType AS TABLE
(
   col1 INT NOT NULL,
   col2 CHAR(10)
)WITH (MEMORY_OPTIMIZED=ON);
```

Глобальная временная таблица выглядит так:

```
CREATE TABLE ##tempGlobalTabel
(
    Col1   INT   NOT NULL ,
    Col2   NVARCHAR(4000)
);
```

Глобальная временная таблица с оптимизацией памяти:

```
CREATE TABLE dbo.tempGlobalTabel
(
    Col1   INT   NOT NULL   INDEX ix NONCLUSTERED,
    Col2   NVARCHAR(4000)
)
    WITH
        (MEMORY_OPTIMIZED = ON,
         DURABILITY = SCHEMA_ONLY);
```

Для оптимизации глобальных глобальных временных таблиц (## temp):

1. Создайте новую таблицу с оптимизацией памяти `SCHEMA_ONLY` с той же схемой, что и глобальная таблица `##temp`
    - Убедитесь, что в новой таблице есть хотя бы один индекс
2. Измените все ссылки на `##temp` в операторах Transact-SQL на новую оптимизированную для памяти таблицу temp
3. Замените операторы `DROP TABLE ##temp` в вашем коде с помощью `DELETE FROM temp` , чтобы очистить содержимое
4. Удалите из своего кода инструкции `CREATE TABLE ##temp` - они теперь избыточны

больше информации

Объявлять переменные таблицы с оптимизацией памяти

Для более высокой производительности вы можете оптимизировать свою таблицу с помощью памяти. Вот T-SQL для традиционной переменной таблицы:

```
DECLARE @tvp TABLE
(
    col1   INT NOT NULL ,
    Col2   CHAR(10)
);
```

Чтобы определить переменные, оптимизированные для памяти, сначала необходимо

создать тип таблицы с оптимизацией памяти, а затем объявить из нее переменную:

```
CREATE TYPE dbo.memTypeTable
AS TABLE
(
    Col1  INT NOT NULL INDEX ix1,
    Col2  CHAR(10)
)
WITH
    (MEMORY_OPTIMIZED = ON);
```

Затем мы можем использовать тип таблицы следующим образом:

```
DECLARE @tvp memTypeTable
insert INTO @tvp
values (1,'1'),(2,'2'),(3,'3'),(4,'4'),(5,'5'),(6,'6')

SELECT * FROM @tvp
```

Результат:

```
Col1    Col2
1       1
2       2
3       3
4       4
5       5
6       6
```

Создать временную таблицу с оптимизированной памятью с системой

```
CREATE TABLE [dbo].[MemOptimizedTemporalTable]
(
    [BusinessDocNo] [bigint] NOT NULL,
    [ProductCode] [int] NOT NULL,
    [UnitID] [tinyint] NOT NULL,
    [PriceID] [tinyint] NOT NULL,
    [SysStartTime] [datetime2](7) GENERATED ALWAYS AS ROW START NOT NULL,
    [SysEndTime] [datetime2](7) GENERATED ALWAYS AS ROW END NOT NULL,
    PERIOD FOR SYSTEM_TIME ([SysStartTime], [SysEndTime]),

    CONSTRAINT [PK_MemOptimizedTemporalTable]  PRIMARY KEY NONCLUSTERED
    (
        [BusinessDocNo] ASC,
        [ProductCode] ASC
    )
)
WITH (
    MEMORY_OPTIMIZED = ON , DURABILITY = SCHEMA_AND_DATA, -- Memory Optimized Option ON
    SYSTEM_VERSIONING = ON (HISTORY_TABLE = [dbo].[MemOptimizedTemporalTable_History] ,
DATA_CONSISTENCY_CHECK = ON )
)
```

больше информации

Прочитайте Встроенная память OLTP (Hekaton) онлайн: https://riptutorial.com/ru/sql-server/topic/5295/встроенная-память-oltp--hekaton-

глава 25: Вычисляемые столбцы

Examples

Столбец вычисляется из выражения

Вычисленный столбец вычисляется из выражения, которое может использовать другие столбцы в одной таблице. Выражение может быть именем неукомплектованного столбца, константой, функцией и любой комбинацией из них, связанной одним или несколькими операторами.

Создать таблицу с вычисленным столбцом

```
Create table NetProfit
(
    SalaryToEmployee            int,
    BonusDistributed            int,
    BusinessRunningCost         int,
    BusinessMaintenanceCost     int,
    BusinessEarnings            int,
    BusinessNetIncome
                As BusinessEarnings - (SalaryToEmployee          +
                                       BonusDistributed          +
                                       BusinessRunningCost       +
                                       BusinessMaintenanceCost    )

)
```

Значение вычисляется и сохраняется в вычисленном столбце автоматически при вставке других значений.

```
Insert Into NetProfit
    (SalaryToEmployee,
     BonusDistributed,
     BusinessRunningCost,
     BusinessMaintenanceCost,
     BusinessEarnings)
Values
    (1000000,
     10000,
     1000000,
     50000,
     2500000)
```

Простой пример, который мы обычно используем в журнальных таблицах

```
CREATE TABLE [dbo].[ProcessLog](
[LogId] [int] IDENTITY(1,1) NOT NULL,
[LogType] [varchar](20) NULL,
[StartTime] [datetime] NULL,
```

```
[EndTime] [datetime] NULL,
[RunMinutes]  AS
(datediff(minute,coalesce([StartTime],getdate()),coalesce([EndTime],getdate())))
```

Это дает разницу хода в минутах для времени выполнения, что будет очень удобно.

Прочитайте Вычисляемые столбцы онлайн: https://riptutorial.com/ru/sql-server/topic/5561/вычисляемые-столбцы

глава 26: ГРУППА ПО

Examples

Простая группировка

Таблица заказов

Пользовательский ИД	Код товара	Количество	Цена
1	2	5	100
1	3	2	200
1	4	1	500
2	1	4	50
3	5	6	700

При группировке по определенному столбцу возвращаются только уникальные значения этого столбца.

```
SELECT customerId
FROM orders
GROUP BY customerId;
```

Возвращаемое значение:

Пользовательский ИД
1
2
3

Совокупные функции, такие как `count()` применяются к каждой группе, а не к полной таблице:

```
SELECT customerId,
       COUNT(productId) as numberOfProducts,
       sum(price) as totalPrice
FROM orders
GROUP BY customerId;
```

Возвращаемое значение:

Пользовательский ИД	numberOfProducts	Итоговая цена
1	3	800
2	1	50
3	1	700

ГРУППА несколькими столбцами

Можно захотеть GROUP BY более чем одним столбцом

```
declare @temp table(age int, name varchar(15))

insert into @temp
select 18, 'matt' union all
select 21, 'matt' union all
select 21, 'matt' union all
select 18, 'luke' union all
select 18, 'luke' union all
select 21, 'luke' union all
select 18, 'luke' union all
select 21, 'luke'

SELECT Age, Name, count(1) count
FROM @temp
GROUP BY Age, Name
```

будут группироваться по возрасту и имени и будут производить:

Возраст	название	подсчитывать
18	Люк	3
21	Люк	2
18	матовый	1
21	матовый	2

Группировать с помощью нескольких таблиц, несколько столбцов

Группа часто используется с заявлением о соединении. Предположим, у нас есть две таблицы. Первая - таблица студентов:

Я бы	ФИО	Возраст
1	Мэтт Джонс	20
2	Frank Blue	21
3	Энтони Ангел	18

Вторая таблица - это таблица темы, которую может принять каждый студент:

subject_id	Предмет
1	математика
2	PE
3	физика

И потому, что один студент может посещать множество предметов, и один предмет может посещать многие ученики (поэтому отношение N: N), нам нужно иметь третью «ограничительную» таблицу. Назовем таблицу Students_subjects:

subject_id	Студенческий билет
1	1
2	2
2	1
3	2
1	3
1	1

Теперь позвольте сказать, что мы хотим знать количество предметов, которые посещают каждый ученик. Здесь автономный оператор `GROUP BY` недостаточен, поскольку информация недоступна через одну таблицу. Поэтому мы должны использовать `GROUP BY` с оператором `JOIN` :

```
Select Students.FullName, COUNT(Subject Id) as SubjectNumber FROM Students_Subjects
LEFT JOIN Students
ON Students_Subjects.Student_id = Students.Id
GROUP BY Students.FullName
```

Результат данного запроса выглядит следующим образом:

ФИО	SubjectNumber
Мэтт Джонс	3
Frank Blue	2
Энтони Ангел	1

Для еще более сложного примера использования GROUP BY, скажем, студент может присвоить одно и то же имя своему имени более одного раза (как показано в таблице Students_Subjects). В этом случае мы могли бы подсчитать количество раз, когда каждый предмет был назначен ученику по GROUPing более чем на один столбец:

```
SELECT Students.FullName, Subjects.Subject,
COUNT(Students_subjects.Subject_id) AS NumberOfOrders
FROM ((Students_Subjects
INNER JOIN Students
ON Students_Subjcets.Student_id=Students.Id)
INNER JOIN Subjects
ON Students_Subjects.Subject_id=Subjects.Subject_id)
GROUP BY Fullname,Subject
```

Этот запрос дает следующий результат:

ФИО	Предмет	SubjectNumber
Мэтт Джонс	математика	2
Мэтт Джонс	PE	1
Frank Blue	PE	1
Frank Blue	физика	1
Энтони Ангел	математика	1

HAVING

Поскольку `WHERE` оценивается до `GROUP BY` , вы не можете использовать `WHERE` для оценки результатов группировки (как правило, агрегированную функцию, такую как `COUNT(*)`). Чтобы удовлетворить эту потребность, можно использовать предложение `HAVING` .

Например, используя следующие данные:

```
DECLARE @orders TABLE(OrderID INT, Name NVARCHAR(100))

INSERT INTO @orders VALUES
( 1, 'Matt' ),
( 2, 'John' ),
```

```
( 3, 'Matt' ),
( 4, 'Luke' ),
( 5, 'John' ),
( 6, 'Luke' ),
( 7, 'John' ),
( 8, 'John' ),
( 9, 'Luke' ),
( 10, 'John' ),
( 11, 'Luke' )
```

Если мы хотим получить количество заказов, которые разместили каждый человек, мы будем использовать

```
SELECT Name, COUNT(*) AS 'Orders'
FROM @orders
GROUP BY Name
```

и получить

название	заказы
Matt	2
Джон	5
Люк	4

Однако, если мы хотим ограничить это для лиц, которые разместили более двух заказов, мы можем добавить предложение `HAVING` .

```
SELECT Name, COUNT(*) AS 'Orders'
FROM @orders
GROUP BY Name
HAVING COUNT(*) > 2
```

даст

название	заказы
Джон	5
Люк	4

Обратите внимание, что, подобно `GROUP BY` , столбцы, помещенные в `HAVING` должны точно соответствовать их аналогам в `SELECT` . Если бы в приведенном выше примере мы сказали, что

```
SELECT Name, COUNT(DISTINCT OrderID)
```

наше предложение `HAVING` должно было бы сказать

```
HAVING COUNT(DISTINCT OrderID) > 2
```

GROUP BY с ROLLUP и CUBE

Оператор ROLLUP полезен при создании отчетов, содержащих промежуточные итоги и итоговые значения.

- CUBE создает набор результатов, который показывает агрегаты для всех комбинаций значений в выбранных столбцах.
- ROLLUP создает набор результатов, который показывает агрегаты для иерархии значений в выбранных столбцах.

Вещь	цвет	Количество
Таблица	синий	124
Таблица	красный	223
кресло	синий	101
кресло	красный	210

```
SELECT CASE WHEN (GROUPING(Item) = 1) THEN 'ALL'
            ELSE ISNULL(Item, 'UNKNOWN')
       END AS Item,
       CASE WHEN (GROUPING(Color) = 1) THEN 'ALL'
            ELSE ISNULL(Color, 'UNKNOWN')
       END AS Color,
       SUM(Quantity) AS QtySum
FROM Inventory
GROUP BY Item, Color WITH ROLLUP

Item                 Color                QtySum
-------------------- -------------------- ---------------------------
Chair                Blue                 101.00
Chair                Red                  210.00
Chair                ALL                  311.00
Table                Blue                 124.00
Table                Red                  223.00
Table                ALL                  347.00
ALL                  ALL                  658.00
```

(Затронуто 7 строк)

Если ключевое слово ROLLUP в запросе будет изменено на CUBE, набор результатов CUBE будет таким же, за исключением того, что в конце будут возвращены эти две дополнительные строки:

```
ALL                  Blue                 225.00
ALL                  Red                  433.00
```

https://technet.microsoft.com/en-us/library/ms189305(v=sql.90).aspx

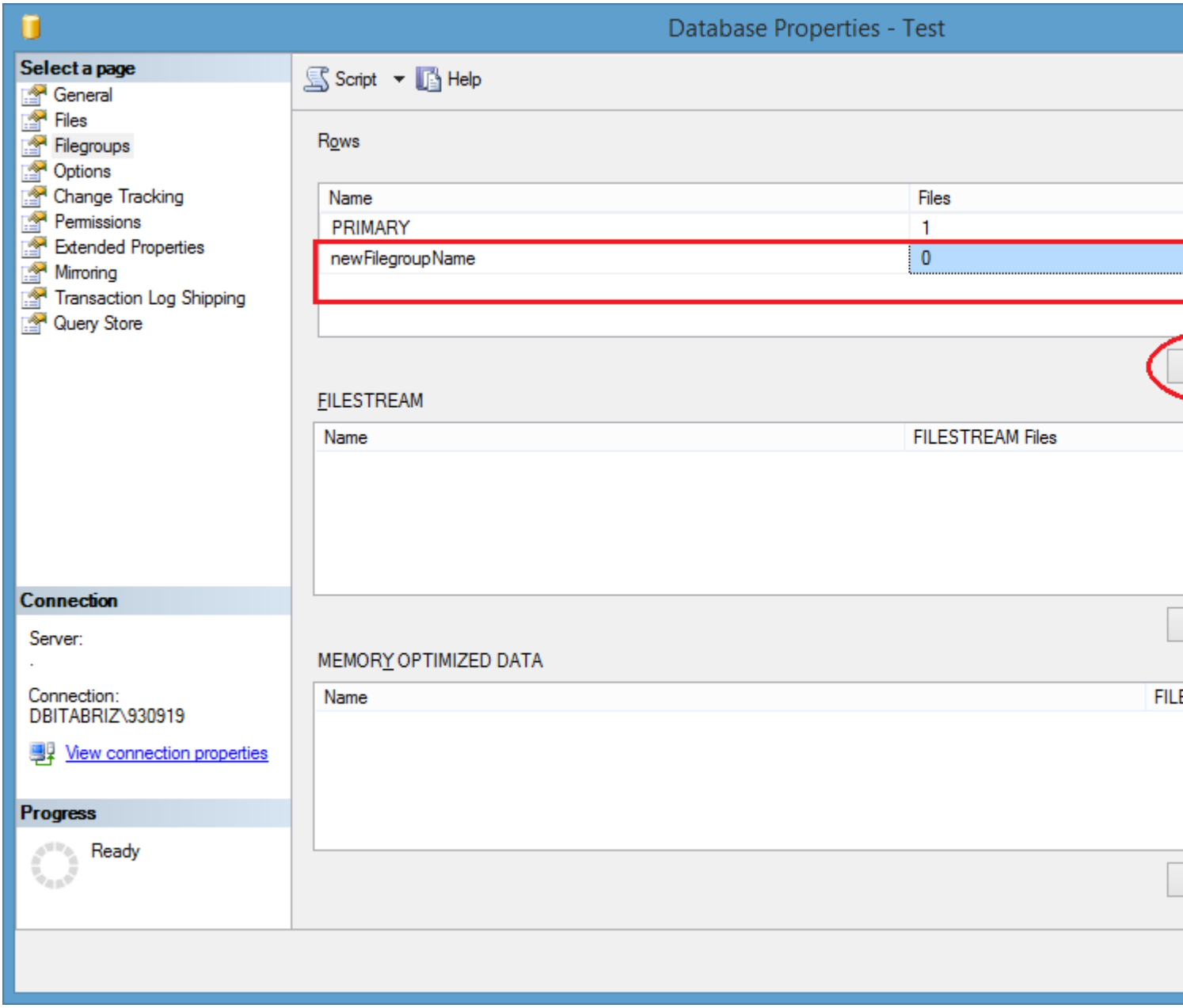
Прочитайте ГРУППА ПО онлайн: https://riptutorial.com/ru/sql-server/topic/3231/группа-по

глава 27: Группа файлов

Examples

Создание файловой группы в базе данных

Мы можем создать его двумя способами. Сначала из режима конструктора свойств базы данных:



И по sql-скриптам:

```
USE master;
GO
-- Create the database with the default data
```

```
-- filegroup and a log file. Specify the
-- growth increment and the max size for the
-- primary data file.

CREATE DATABASE TestDB ON PRIMARY
(
    NAME = 'TestDB_Primary',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\TestDB_Prm.mdf',
    SIZE = 1 GB,
    MAXSIZE = 10 GB,
    FILEGROWTH = 1 GB
), FILEGROUP TestDB_FG1
(
    NAME = 'TestDB_FG1_1',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\TestDB_FG1_1.ndf',
    SIZE = 10 MB,
    MAXSIZE = 10 GB,
    FILEGROWTH = 1 GB
),
(
    NAME = 'TestDB_FG1_2',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\TestDB_FG1_2.ndf',
    SIZE = 10 MB,
    MAXSIZE = 10 GB,
    FILEGROWTH = 1 GB
) LOG ON
(
    NAME = 'TestDB_log',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\TestDB.ldf',
    SIZE = 10 MB,
    MAXSIZE = 10 GB,
    FILEGROWTH = 1 GB
);

go
ALTER DATABASE TestDB MODIFY FILEGROUP TestDB_FG1 DEFAULT;
go

-- Create a table in the user-defined filegroup.
USE TestDB;
Go

CREATE TABLE MyTable
(
    col1 INT PRIMARY KEY,
    col2 CHAR(8)
)
ON TestDB_FG1;
GO
```

Прочитайте Группа файлов онлайн: https://riptutorial.com/ru/sql-server/topic/5461/группа-файлов

глава 28: Губернатор ресурсов

замечания

Resource Governor в SQL Server - это функция, которая позволяет вам управлять использованием ресурсов различными приложениями и пользователями. Он запускается в реальном времени, устанавливая пределы ЦП и памяти. Это поможет предотвратить то, что один тяжелый процесс будет потреблять все системные ресурсы, в то время как к ним ждут более мелкие задачи.

Доступно только в Enterprise Editions

Examples

Чтение статистики

```
select *
from sys.dm_resource_governor_workload_groups

select *
from sys.dm_resource_governor_resource_pools
```

Создание пула для adhoc-запросов

Сначала создайте пул ресурсов, кроме стандартного

```
CREATE RESOURCE POOL [PoolAdhoc] WITH(min_cpu_percent=0,
        max_cpu_percent=50,
        min_memory_percent=0,
        max_memory_percent=50)
GO
```

Создание рабочей группы для пула

```
CREATE WORKLOAD GROUP [AdhocMedium] WITH(importance=Medium) USING [PoolAdhoc]
```

Создайте функцию, которая содержит логику для регулятора ресурса и приложите его

```
create function [dbo].[ufn_ResourceGovernorClassifier]()
  returns sysname with schemabinding
as
begin
    return CASE
                WHEN APP_NAME() LIKE 'Microsoft Office%'                        THEN
'AdhocMedium'        -- Excel
                WHEN APP_NAME() LIKE 'Microsoft SQL Server Management Studio%'    THEN
'AdhocMedium'        -- Adhoc SQL
```

```
                WHEN SUSER_NAME() LIKE 'DOMAIN\username'                    THEN 'AdhocMedium'
-- Ssis
                ELSE 'default'
            END
end

GO

alter resource governor
with (classifier_function = dbo.ufn_ResourceGovernorClassifier)

GO

alter resource governor reconfigure

GO
```

Прочитайте Губернатор ресурсов онлайн: https://riptutorial.com/ru/sql-server/topic/4146/губернатор-ресурсов

глава 29: Даты

Синтаксис

- КОНМЕСЯЦА *(дата_начала* [, month_to_add])

замечания

согласно https://msdn.microsoft.com/en-us/library/ms187819.aspx , `DateTime` s являются точными до 3 мс.

Округление данных datetime. Дробные значения секунды. Точность datetime округляется до приращений .000, .003 или .007 секунд, как показано в следующей таблице.

Пользовательское значение	Системное сохраненное значение
01/01/98 23: 59: 59.999	1998-01-02 00: 00: 00.000
------	------
01/01/98 23: 59: 59.995	1998-01-01 23: 59: 59.997
01/01/98 23: 59: 59.996	
01/01/98 23: 59: 59.997	
01/01/98 23: 59: 59.998	
------	------
01/01/98 23: 59: 59.992	1998-01-01 23: 59: 59.993
01/01/98 23: 59: 59.993	
01/01/98 23: 59: 59.994	
------	------
01/01/98 23: 59: 59.990	1998-01-01 23: 59: 59.990
01/01/98 23: 59: 59.991	
------	------

Если требуется более высокая точность, следует использовать `time` , `datetime2` или `datetimeoffset` .

Examples

Форматирование даты и времени с использованием CONVERT

Вы можете использовать функцию CONVERT для приведения типа данных datetime в форматированную строку.

```
SELECT GETDATE() AS [Result] -- 2016-07-21 07:56:10.927
```

Вы можете также использовать некоторые встроенные коды для преобразования в определенный формат. Вот варианты, встроенные в SQL Server:

```
DECLARE @convert_code INT = 100 -- See Table Below
SELECT CONVERT(VARCHAR(30), GETDATE(), @convert_code) AS [Result]
```

@convert_code	Результат
100	"21 июля 2016 7:56 утра"
101	"07/21/2016"
102	"2016.07.21"
103	"21/07/2016"
104	"21.07.2016"
105	"21-07-2016"
106	«21 июля 2016 года»
107	«21 июля 2016 года»
108	«7:57:05»
109	"21 июля 2016 7: 57: 45: 707AM"
110	"07-21-2016"
111	"2016/07/21"
112	"20160721"
113	"21 июля 2016 года 07: 57: 59: 553"
114	"07: 57: 59: 553"
120	«2016-07-21 07:57:59»

@convert_code	Результат
121	"2016-07-21 07: 57: 59.553"
126	"2016-07-21T07: 58: 34,340"
127	"2016-07-21T07: 58: 34,340"
130	"16: 1437 7: 58: 34: 340AM"
131	"16/10/1437 7: 58: 34: 340AM"

```
SELECT GETDATE() AS [Result]                                -- 2016-07-21 07:56:10.927
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),100) AS [Result] -- Jul 21 2016  7:56AM
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),101) AS [Result] -- 07/21/2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),102) AS [Result] -- 2016.07.21
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),103) AS [Result] -- 21/07/2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),104) AS [Result] -- 21.07.2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),105) AS [Result] -- 21-07-2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),106) AS [Result] -- 21 Jul 2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),107) AS [Result] -- Jul 21, 2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),108) AS [Result] -- 07:57:05
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),109) AS [Result] -- Jul 21 2016  7:57:45:707AM
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),110) AS [Result] -- 07-21-2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),111) AS [Result] -- 2016/07/21
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),112) AS [Result] -- 20160721
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),113) AS [Result] -- 21 Jul 2016 07:57:59:553
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),114) AS [Result] -- 07:57:59:553
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),120) AS [Result] -- 2016-07-21 07:57:59
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),121) AS [Result] -- 2016-07-21 07:57:59.553
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),126) AS [Result] -- 2016-07-21T07:58:34.340
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),127) AS [Result] -- 2016-07-21T07:58:34.340
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),130) AS [Result] -- 16 ???? 1437  7:58:34:340AM
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),131) AS [Result] -- 16/10/1437  7:58:34:340AM
```

Форматирование даты и времени с использованием FORMAT

SQL Server 2012

Вы можете использовать новую функцию: `FORMAT()` .

Используя это, вы можете преобразовать поля `DATETIME` в свой собственный формат `VARCHAR` .

пример

```
DECLARE @Date DATETIME = '2016-09-05 00:01:02.333'

SELECT FORMAT(@Date, N'dddd, MMMM dd, yyyy hh:mm:ss tt')
```

Понедельник, 5 сентября 2016 года 12:01:02

аргументы

Учитывая, что формат `DATETIME` отформатирован, `2016-09-05 00:01:02.333` , следующий график показывает, каков будет их вывод для предоставленного аргумента.

аргументация	Выход
гггг	2016
уу	16
ММММ	сентябрь
М.М.	09
М	9
дддд	понедельник
ддд	понедельник
дд	05
d	5
НН	00
ЧАС	0
чч	12
час	12
мм	01
м	1
сс	02
s	2
тт	AM
Т	
FFF	333
Ф.Ф.	33
е	3

Вы также можете предоставить один аргумент функции `FORMAT()` для генерации предварительно форматированного вывода:

```
DECLARE @Date DATETIME = '2016-09-05 00:01:02.333'

SELECT FORMAT(@Date, N'U')
```

Понедельник, 5 сентября 2016 года 4:01:02

Одиночный аргумент	Выход
D	Понедельник, 5 сентября 2016 г.
d	9/5/2016
F	Понедельник, 5 сентября 2016 года 12:01:02
e	Понедельник, 5 сентября 2016 года 12:01
г	5/5/2016 12:01:02
г	5/5/2016 12:01
M	Сентябрь 05
O	2016-09-05T00: 01: 02,3330000
p	Пн, 05 сен 2016 00:01:02 GMT
s	2016-09-05T00: 01: 02
T	12:01:02 AM
T	12:01
U	Понедельник, 5 сентября 2016 года 4:01:02
U	2016-09-05 00: 01: 02Z
Y	Сентябрь 2016 года

Примечание. В приведенном выше списке используется культура `en-US` . Для `FORMAT()` через третий параметр можно указать другую культуру:

```
DECLARE @Date DATETIME = '2016-09-05 00:01:02.333'

SELECT FORMAT(@Date, N'U', 'zh-cn')
```

2016 9 5 4:01:02

Получить текущую дату

Встроенные функции `GETDATE` и `GETUTCDATE` возвращают текущую дату и время без смещения часового пояса.

Возвращаемое значение обеих функций основано на операционной системе компьютера, на котором запущен экземпляр SQL Server.

Возвращаемое значение GETDATE представляет текущее время в том же часовом поясе, что и операционная система. Возвращаемое значение GETUTCDATE представляет текущее время UTC.

Любая функция может быть включена в предложение `SELECT` запроса или как часть логического выражения в `WHERE` .

Примеры:

```
-- example query that selects the current time in both the server time zone and UTC
SELECT GETDATE() as SystemDateTime, GETUTCDATE() as UTCDateTime

-- example query records with EventDate in the past.
SELECT * FROM MyEvents WHERE EventDate < GETDATE()
```

Есть еще несколько встроенных функций, которые возвращают разные варианты текущего времени:

```
SELECT
    GETDATE(),          --2016-07-21 14:27:37.447
    GETUTCDATE(),       --2016-07-21 18:27:37.447
    CURRENT_TIMESTAMP,  --2016-07-21 14:27:37.447
    SYSDATETIME(),      --2016-07-21 14:27:37.4485768
    SYSDATETIMEOFFSET(),--2016-07-21 14:27:37.4485768 -04:00
    SYSUTCDATETIME()    --2016-07-21 18:27:37.4485768
```

DATEADD для добавления и вычитания периодов времени

Общий синтаксис:

```
DATEADD (datepart , number , datetime_expr)
```

Чтобы добавить меру времени, `number` должен быть положительным. Чтобы вычесть значение времени, `number` должно быть отрицательным.

Примеры

```
DECLARE @now DATETIME2 = GETDATE();
SELECT @now;                        --2016-07-21 14:39:46.4170000
SELECT DATEADD(YEAR, 1, @now)       --2017-07-21 14:39:46.4170000
SELECT DATEADD(QUARTER, 1, @now)    --2016-10-21 14:39:46.4170000
```

```
SELECT DATEADD(WEEK, 1, @now)       --2016-07-28 14:39:46.4170000
SELECT DATEADD(DAY, 1, @now)        --2016-07-22 14:39:46.4170000
SELECT DATEADD(HOUR, 1, @now)       --2016-07-21 15:39:46.4170000
SELECT DATEADD(MINUTE, 1, @now)     --2016-07-21 14:40:46.4170000
SELECT DATEADD(SECOND, 1, @now)     --2016-07-21 14:39:47.4170000
SELECT DATEADD(MILLISECOND, 1, @now)--2016-07-21 14:39:46.4180000
```

ПРИМЕЧАНИЕ. `DATEADD` также принимает аббревиатуры в параметре `datepart` . Использование этих сокращений обычно обескураживается, поскольку они могут вводить в заблуждение (`m` vs `mi` , `ww` vs `w` и т. Д.).

Ссылка на детали даты

Это значения `datepart` доступные для функций даты и времени:

DatePart	Сокращения
год	yy, yyyy
четверть	qq, q
месяц	мм, м
DayOfYear	dy, y
день	dd, d
неделю	wk, ww
будний день	dw, w
час	чч
минут	mi, n
второй	ss, s
миллисекунды	Миз
микросекунда	MCS
наносекунда	нс

ПРИМЕЧАНИЕ . Использование сокращений обычно обескураживается, поскольку они могут вводить в заблуждение (`m` vs `mi` , `ww` vs `w` и т. Д.). Длинная версия представления `datepart` способствует ясности и удобочитаемости и должна использоваться по возможности (`month` , `minute` , `week` , `weekday` и т. Д.).

DATEDIFF для расчета разницы во времени

Общий синтаксис:

```
DATEDIFF (datepart, datetime_expr1, datetime_expr2)
```

Он вернет положительное число, если `datetime_expr` в прошлом `datetime_expr2` к `datetime_expr2` и отрицательное число в противном случае.

Примеры

```
DECLARE @now DATETIME2 = GETDATE();
DECLARE @oneYearAgo DATETIME2 = DATEADD(YEAR, -1, @now);
SELECT @now                                  --2016-07-21 14:49:50.9800000
SELECT @oneYearAgo                           --2015-07-21 14:49:50.9800000
SELECT DATEDIFF(YEAR, @oneYearAgo, @now)     --1
SELECT DATEDIFF(QUARTER, @oneYearAgo, @now)  --4
SELECT DATEDIFF(WEEK, @oneYearAgo, @now)     --52
SELECT DATEDIFF(DAY, @oneYearAgo, @now)      --366
SELECT DATEDIFF(HOUR, @oneYearAgo, @now)     --8784
SELECT DATEDIFF(MINUTE, @oneYearAgo, @now)   --527040
SELECT DATEDIFF(SECOND, @oneYearAgo, @now)   --31622400
```

ПРИМЕЧАНИЕ. `DATEDIFF` также принимает аббревиатуры в параметре `datepart` . Использование этих сокращений обычно обескураживается, поскольку они могут вводить в заблуждение (`m` vs `mi` , `ww` vs `w` и т. Д.).

`DATEDIFF` также может использоваться для определения смещения между UTC и локальным временем SQL Server. Следующий оператор можно использовать для расчета смещения между временем по Гринвичу и местным временем (включая часовой пояс).

```
select  DATEDIFF(hh, getutcdate(), getdate()) as 'CentralTimeOffset'
```

DATEPART & DATENAME

`DATEPART` возвращает указанное число `datepart` указанного выражения datetime как числовое значение.

`DATENAME` возвращает строку символов, которая представляет заданную `datepart` в определенную дату. На практике `DATENAME` в основном полезен для получения названия месяца или дня недели.

Существуют также некоторые сокращенные функции для получения года, месяца или дня выражения datetime, которые ведут себя как `DATEPART` с их соответствующими единицами `datepart` .

Синтаксис:

```
DATEPART ( datepart , datetime_expr )
DATENAME ( datepart , datetime_expr )
DAY ( datetime_expr )
MONTH ( datetime_expr )
YEAR ( datetime_expr )
```

Примеры:

```
DECLARE @now DATETIME2 = GETDATE();
SELECT @now                       --2016-07-21 15:05:33.8370000
SELECT DATEPART(YEAR, @now)       --2016
SELECT DATEPART(QUARTER, @now)    --3
SELECT DATEPART(WEEK, @now)       --30
SELECT DATEPART(HOUR, @now)       --15
SELECT DATEPART(MINUTE, @now)     --5
SELECT DATEPART(SECOND, @now)     --33
-- Differences between DATEPART and DATENAME:
SELECT DATEPART(MONTH, @now)      --7
SELECT DATENAME(MONTH, @now)      --July
SELECT DATEPART(WEEKDAY, @now)    --5
SELECT DATENAME(WEEKDAY, @now)    --Thursday
--shorthand functions
SELECT DAY(@now)    --21
SELECT MONTH(@now)  --7
SELECT YEAR(@now)   --2016
```

ПРИМЕЧАНИЕ. `DATEPART` и `DATENAME` также принимают аббревиатуры в параметре `datepart` . Использование этих сокращений обычно обескураживается, поскольку они могут вводить в заблуждение (`m` vs `mi` , `ww` vs `w` и т. Д.).

Получение последнего дня месяца

Используя функции `DATEADD` и `DATEDIFF` , можно вернуть последнюю дату месяца.

```
SELECT DATEADD(d, -1, DATEADD(m, DATEDIFF(m, 0, '2016-09-23') + 1, 0))
-- 2016-09-30 00:00:00.000
```

SQL Server 2012

Функция `EOMONTH` обеспечивает более сжатый способ возврата последней даты месяца и имеет необязательный параметр для смещения месяца.

```
SELECT EOMONTH('2016-07-21')        --2016-07-31
SELECT EOMONTH('2016-07-21', 4)     --2016-11-30
SELECT EOMONTH('2016-07-21', -5)    --2016-02-29
```

Вернуть только дату из DateTime

Существует много способов вернуть дату из объекта DateTime

1. `SELECT CONVERT(Date, GETDATE())`
2. `SELECT DATEADD(dd, 0, DATEDIFF(dd, 0, GETDATE()))` возвращается 2016-07-21 00: 00: 00.000

3. `SELECT CAST(GETDATE() AS DATE)`
4. `SELECT CONVERT(CHAR(10),GETDATE(),111)`
5. `SELECT FORMAT(GETDATE(), 'yyyy-MM-dd')`

Обратите внимание, что опции 4 и 5 возвращают строку, а не дату.

Создать функцию для расчета возраста человека на определенную дату

Эта функция будет принимать 2 параметра datetime, DOB и дату для проверки возраста на

```
  CREATE FUNCTION [dbo].[Calc_Age]
    (
    @DOB datetime , @calcDate datetime
    )
    RETURNS int
    AS
   BEGIN
declare @age int

IF (@calcDate < @DOB  )
RETURN -1

-- If a DOB is supplied after the comparison date, then return -1
SELECT @age = YEAR(@calcDate) - YEAR(@DOB) +
  CASE WHEN DATEADD(year,YEAR(@calcDate) - YEAR(@DOB)
  ,@DOB) > @calcDate THEN -1 ELSE 0 END


RETURN @age


END
```

например, чтобы проверить возраст сегодня кого-то, родившегося 1 / 1/2000

```
SELECT  dbo.Calc_Age('2000-01-01',Getdate())
```

ОБЪЕКТ ПРОГРАММЫ CROSS PLATFORM

SQL Server 2012

В Transact SQL вы можете определить объект как `Date` (или `DateTime`), используя `[DATEFROMPARTS][1]` (или `[DATETIMEFROMPARTS][1]`), например, следующую:

```
 DECLARE @myDate DATE=DATEFROMPARTS(1988,11,28)
 DECLARE @someMoment DATETIME=DATEFROMPARTS(1988,11,28,10,30,50,123)
```

Параметры, которые вы предоставляете: Год, Месяц, День для функции `DATEFROMPARTS` а для функции `DATETIMEFROMPARTS` вам необходимо `DATEFROMPARTS` год, месяц, день, час, минуты, секунды и миллисекунды.

Эти методы полезны и заслуживают внимания, потому что использование простой строки

для создания даты (или даты и времени) может привести к сбою в зависимости от настроек региона, местоположения или формата файла главной машины.

Формат даты расширенный

Формат даты	Заявление SQL	Образец вывода
YY-MM-DD	SELECT RIGHT (CONVERT (VARCHAR (10), SYSDATETIME (), 20), 8) AS [YY-MM-DD] SELECT REPLACE (CONVERT (VARCHAR (8), SYSDATETIME (), 11), '/', '-') AS [YY-MM-DD]	11-06-08
YYYY-MM-DD	SELECT CONVERT (VARCHAR (10), SYSDATETIME (), 120) AS [YYYY-MM-DD] SELECT REPLACE (CONVERT (VARCHAR (10), SYSDATETIME (), 111), '/', '-') AS [YYYY-MM-DD]	2011-06-08
YYYY-MD	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) AS [YYYY-MD]	2011-6-8
YY-MD	SELECT RIGHT (CAST (YAAR (SYSDATETIME ()) AS VARCHAR (4)), 2) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (ДЕНЬ (SYSDATETIME ( )) AS VARCHAR (2)) AS [YY-MD]	11-6-8
MD-YYYY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [MD-YYYY]	6-8-2011
MD-YY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '-' + RIGHT (CAST (YAAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [MD-YY]	6-8-11
DM-YYYY	SELECT CAST (ДЕНЬ (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [DM-YYYY]	8-6-2011
DM-YY	SELECT CAST (ДЕНЬ (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + RIGHT (CAST (YAAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [DM-YY]	8-6-11

Формат даты	Заявление SQL	Образец вывода
YY-MM	SELECT RIGHT (CONVERT (VARCHAR (7), SYSDATETIME (), 20), 5) AS [YY-MM] SELECT SUBSTRING (CONVERT (VARCHAR (10), SYSDATETIME (), 120), 3, 5) AS [YY-MM]	11-06
YYYY-MM	SELECT CONVERT (VARCHAR (7), SYSDATETIME (), 120) AS [YYYY-MM]	2011-06
YY-M	SELECT RIGHT (CAST (YAAR (SYSDATETIME ()) AS VARCHAR (4)), 2) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [YY-M]	11-6
YYYY-M	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [YYYY-M]	2011-6
MM-YY	SELECT RIGHT (CONVERT (VARCHAR (8), SYSDATETIME (), 5), 5) AS [MM-YY] SELECT SUBSTRING (CONVERT (VARCHAR (8), SYSDATETIME (), 5), 4, 5) AS [MM-YY]	06-11
MM-YYYY	SELECT RIGHT (CONVERT (VARCHAR (10), SYSDATETIME (), 105), 7) AS [MM-YYYY]	06-2011
M-YY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + RIGHT (CAST (YAAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [M-YY]	6-11
M-YYYY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [M-YYYY]	6-2011
MM-DD	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 10) AS [MM-DD]	06-08
DD-MM	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 5) AS [DD-MM]	08-06
Мэриленд	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (ДЕНЬ (SYSDATETIME ()) AS VARCHAR (2)) AS [MD]	6-8
DM	SELECT CAST (ДЕНЬ (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [DM]	8-6

Формат даты	Заявление SQL	Образец вывода
M / D / YYYY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [M / D / YYYY]	6/8/2011
M / Д / ГГ	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '/' + RIGHT (CAST (YAAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [M / D / YY]	6/8/11
Д / M / YYYY	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [D / M / YYYY]	8/6/2011
Д / M / ГГ	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + RIGHT (CAST (YAAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [D / M / YY]	8/6/11
YYYY / M / Д	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (ДЕНЬ (SYSDATETIME ()) AS VARCHAR (2)) AS [YYYY / M / D]	2011/6/8
ГГ / M / Д	SELECT RIGHT (CAST (YAAR (SYSDATETIME ()) AS VARCHAR (4)), 2) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (ДЕНЬ (SYSDATETIME ( )) AS VARCHAR (2)) AS [YY / M / D]	11/6/8
MM / YY	SELECT RIGHT (CONVERT (VARCHAR (8), SYSDATETIME (), 3), 5) AS [MM / YY]	06/11
MM / YYYY	SELECT RIGHT (CONVERT (VARCHAR (10), SYSDATETIME (), 103), 7) AS [MM / YYYY]	06/2011
M / ГГ	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + RIGHT (CAST (YAAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [M / YY]	6/11
M / YYYY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [M / YYYY]	6/2011
YY / MM	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 11) AS [YY / MM]	11/06

Формат даты	Заявление SQL	Образец вывода
YYYY / MM	SELECT CONVERT (VARCHAR (7), SYSDATETIME (), 111) AS [YYYY / MM]	2011/06
ГГ / M	SELECT RIGHT (CAST (YAAR (SYSDATETIME ()) AS VARCHAR (4)), 2) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [YY / M]	11/6
YYYY / M	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [YYYY / M]	2011/6
MM / DD	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 1) AS [MM / DD]	06/08
DD / MM	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 3) AS [DD / MM]	08/06
M / Д	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (ДЕНЬ (SYSDATETIME ()) AS VARCHAR (2)) AS [M / D]	6/8
Д / M	SELECT CAST (ДЕНЬ (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [D / M]	8/6
MM.DD.YYYY	SELECT REPLACE (CONVERT (VARCHAR (10), SYSDATETIME (), 101), '/', '.') AS [MM.DD.YYYY]	06.08.2011
MM.DD.YY	SELECT REPLACE (CONVERT (VARCHAR (8), SYSDATETIME (), 1), '/', '.') AS [MM.DD.YY]	06.08.11
MDYYYY	SELECT CAST (МЕСЯЦ (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (ДЕНЬ (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [MDYYYY]	6.8.2011
MDYY	SELECT CAST (МЕСЯЦ (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (ДЕНЬ (SYSDATETIME ()) AS VARCHAR (2)) + '.' + ПРАВО (CAST (ГОД (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [MDYY]	6.8.11
ДД.ММ.ГГГГ	SELECT CONVERT (VARCHAR (10), SYSDATETIME (), 104) AS [DD.MM.YYYY]	08.06.2011
DD.MM.YY	SELECT CONVERT (VARCHAR (10), SYSDATETIME (), 4) AS [DD.MM.YY]	08.06.11

Формат даты	Заявление SQL	Образец вывода
DMYYYY	SELECT CAST (ДЕНЬ (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [DMYYYY]	8.6.2011
DMYY	SELECT CAST (ДЕНЬ (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + ПРАВО (CAST (ГОД (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [DMYY]	8.6.11
YYYY.MD	SELECT CAST (ГОД (SYSDATETIME ()) AS VARCHAR (4)) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (ДЕНЬ (SYSDATETIME ()) AS VARCHAR (2)) AS [YYYY.MD]	2011.6.8
YY.MD	SELECT RIGHT (CAST (ГОД (SYSDATETIME ()) AS VARCHAR (4)), 2) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (ДЕНЬ (SYSDATETIME ()) AS VARCHAR (2)) AS [YY.MD]	11.6.8
MM.YYYY	SELECT RIGHT (CONVERT (VARCHAR (10), SYSDATETIME (), 104), 7) AS [MM.YYYY]	06,2011
мм.гг	SELECT RIGHT (CONVERT (VARCHAR (8), SYSDATETIME (), 4), 5) AS [MM.YY]	06,11
M.YYYY	SELECT CAST (МЕСЯЦ (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [M.YYYY]	6,2011
M.YY	SELECT CAST (МЕСЯЦ (SYSDATETIME ()) AS VARCHAR (2)) + '.' + ПРАВО (CAST (ГОД (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [M.YY]	6,11
YYYY.MM	SELECT CONVERT (VARCHAR (7), SYSDATETIME (), 102) AS [YYYY.MM]	2011,06
YY.MM	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 2) AS [YY.MM]	11,06
YYYY.M	SELECT CAST (ГОД (SYSDATETIME ()) AS VARCHAR (4)) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [YYYY.M]	2011,6
YY.M	SELECT RIGHT (CAST (ГОД (SYSDATETIME ()) AS	11,6

Формат даты	Заявление SQL	Образец вывода
	VARCHAR (4)), 2) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [YY.M]	
MM.DD	SELECT RIGHT (CONVERT (VARCHAR (8), SYSDATETIME (), 2), 5) AS [MM.DD]	06,08
DD.MM	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 4) AS [DD.MM]	08,06
ММДДГГГГ	SELECT REPLACE (CONVERT (VARCHAR (10), SYSDATETIME (), 101), '/', '') AS [MMDDYYYY]	06082011
MMDDYY	SELECT REPLACE (CONVERT (VARCHAR (8), SYSDATETIME (), 1), '/', '') AS [MMDDYY]	060811
DDMMYYYY	SELECT REPLACE (CONVERT (VARCHAR (10), SYSDATETIME (), 103), '/', '') AS [DDMMYYYY]	08062011
DDMMYY	SELECT REPLACE (CONVERT (VARCHAR (8), SYSDATETIME (), 3), '/', '') AS [DDMMYY]	080611
MMYYYY	SELECT RIGHT (REPLACE (CONVERT (VARCHAR (10), SYSDATETIME (), 103), '/', ''), 6) AS [MMYYYY]	062011
MMYY	SELECT RIGHT (REPLACE (CONVERT (VARCHAR (8), SYSDATETIME (), 3), '/', ''), 4) AS [MMYY]	0611
ГГГГММ	SELECT CONVERT (VARCHAR (6), SYSDATETIME (), 112) AS [YYYYMM]	201106
YYMM	SELECT CONVERT (VARCHAR (4), SYSDATETIME (), 12) AS [YYMM]	1106
Месяц DD, ГГГГ	SELECT DATENAME (MONTH, SYSDATETIME ()) + '' + RIGHT ('0' + DATENAME (DAY, SYSDATETIME ()), 2) + ',' + DATENAME (YEAR, SYSDATETIME ()) AS [Month DD, YYYY]	08 июня 2011 г.
Пн ГГГГ	SELECT LEFT (DATENAME (MONTH, SYSDATETIME ()), 3) + '' + DATENAME (YEAR, SYSDATETIME ()) AS [Mon YYYY]	Июн 2011
Месяц ГГГГ	SELECT DATENAME (MONTH, SYSDATETIME ()) + '' + DATENAME (YEAR, SYSDATETIME ()) AS [Месяц ГГГГ]	Июнь 2011 г.
Месяц DD	SELECT RIGHT ('0' + DATENAME (DAY, SYSDATETIME ()), 2) + '' + DATENAME (MONTH, SYSDATETIME ()) AS	08 июня

Формат даты	Заявление SQL	Образец вывода
	[DD Month]	
Месяц DD	SELECT DATENAME (MONTH, SYSDATETIME ()) + '' + RIGHT ('0' + DATENAME (DAY, SYSDATETIME ()), 2) AS [Month DD]	08 июня
DD Месяц YY	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '' + DATENAME (MM, SYSDATETIME ()) + '' + RIGHT (CAST (YAAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [месяц DDYY]	08 июня 11
DD Месяц ГГГГ	SELECT RIGHT ('0' + DATENAME (DAY, SYSDATETIME ()), 2) + '' + DATENAME (MONTH, SYSDATETIME ()) + '' + DATENAME (YEAR, SYSDATETIME ()) AS [DD Month YYYY]	08 июня 2011 г.
Пн-YY	SELECT REPLACE (ПРАВО (КОНВЕРТ (VARCHAR (9), SYSDATETIME (), 6), 6), '', '-') AS [Mon-YY]	Июнь-08
Пн-YYYY	ВЫБРАТЬ ЗАМЕНИТЬ (ПРАВО (КОНВЕРТ (VARCHAR (11), SYSDATETIME (), 106), 8), '', '-') AS [Mon-YYYY]	Июнь-2011
DD-пн-YY	SELECT REPLACE (CONVERT (VARCHAR (9), SYSDATETIME (), 6), '', '-') AS [DD-Mon-YY]	08-Jun-11
DD-YYYY пн-	SELECT REPLACE (CONVERT (VARCHAR (11), SYSDATETIME (), 106), '', '-') AS [DD-Mon-YYYY]	08-Июнь-2011

Прочитайте Даты онлайн: https://riptutorial.com/ru/sql-server/topic/1471/даты

глава 30: Динамический SQL

Examples

Выполнить инструкцию SQL в виде строки

В некоторых случаях вам необходимо выполнить SQL-запрос, помещенный в строку. EXEC, EXECUTE или системная процедура sp_executesql может выполнять любой SQL-запрос, предоставляемый как строка:

```
sp_executesql N'SELECT * FROM sys.objects'
-- or
sp_executesql @stmt = N'SELECT * FROM sys.objects'
-- or
EXEC sp_executesql N'SELECT * FROM sys.objects'
-- or
EXEC('SELECT * FROM sys.columns')
-- or
EXECUTE('SELECT * FROM sys.tables')
```

Эта процедура вернет тот же результат, что и SQL-запрос, предоставленный в виде текста инструкции. sp_executesql может выполнять SQL-запрос, предоставляемый как строковый литерал, переменная / параметр или даже выражение:

```
declare @table nvarchar(40) = N'product items'
EXEC(N'SELECT * FROM ' + @table)
declare @sql nvarchar(40) = N'SELECT * FROM ' + QUOTENAME(@table);
EXEC sp_executesql @sql
```

Вам нужна функция QUOTENAME для вызова специальных символов в переменной @table. Без этой функции вы получите синтаксическую ошибку, если переменная @table содержит что-то вроде пробелов, скобок или любого другого специального символа.

Динамический SQL, выполняемый как пользователь

Вы можете выполнять SQL-запрос как другой пользователь, используя AS USER = 'имя пользователя базы данных'

```
EXEC(N'SELECT * FROM product') AS USER = 'dbo'
```

SQL-запрос будет выполняться под пользователем базы данных dbo. Все проверки разрешений, применимые к пользователю dbo, будут проверяться в SQL-запросе.

SQL Injection с динамическим SQL

Динамические запросы

```
SET @sql = N'SELECT COUNT(*) FROM AppUsers WHERE Username = ''' + @user + ''' AND Password =
''' + @pass + ''''
EXEC(@sql)
```

Если значением переменной пользователя является ***myusername '' OR 1 = 1 -*** будет выполнен следующий запрос:

```
SELECT COUNT(*)
FROM AppUsers
WHERE Username = 'myusername' OR 1=1 --' AND Password = ''
```

Комментарий в конце значения переменной @username будет закомментировать конечную часть запроса и условие 1 = 1 будет оценено. Приложение, которое проверяет его там, по крайней мере, на одного пользователя, возвращаемого этим запросом, вернет счет больше 0, и логин будет успешным.

Используя этот подход, злоумышленник может войти в приложение, даже если он не знает правильного имени пользователя и пароля.

Динамический SQL с параметрами

Чтобы избежать проблем с впрыском и экранированием, динамические SQL-запросы должны выполняться с параметрами, например:

```
SET @sql = N'SELECT COUNT(*) FROM AppUsers WHERE Username = @user AND Password = @pass
EXEC sp_executesql @sql, '@user nvarchar(50), @pass nvarchar(50)', @username, @password
```

Второй параметр - это список параметров, используемых в запросе с их типами, после того, как этот список содержит переменные, которые будут использоваться в качестве значений параметров.

sp_executesql избежит специальных символов и выполнит sql-запрос.

Прочитайте Динамический SQL онлайн: https://riptutorial.com/ru/sql-server/topic/6871/динамический-sql

глава 31: Динамический SQL Pivot

Вступление

В этом разделе описывается, как сделать динамический стержень в SQL Server.

Examples

Базовый динамический SQL Pivot

```
if object_id('tempdb.dbo.#temp') is not null drop table #temp
create table #temp
(
    dateValue datetime,
    category varchar(3),
    amount decimal(36,2)
)

insert into #temp values ('1/1/2012', 'ABC', 1000.00)
insert into #temp values ('2/1/2012', 'DEF', 500.00)
insert into #temp values ('2/1/2012', 'GHI', 800.00)
insert into #temp values ('2/10/2012', 'DEF', 700.00)
insert into #temp values ('3/1/2012', 'ABC', 1100.00)


DECLARE
    @cols AS NVARCHAR(MAX),
    @query  AS NVARCHAR(MAX);

SET @cols = STUFF((SELECT distinct ',' + QUOTENAME(c.category)
            FROM #temp c
            FOR XML PATH(''), TYPE
            ).value('.', 'NVARCHAR(MAX)')
        ,1,1,'')

set @query = '
            SELECT
                dateValue,
                ' + @cols + '
            from
            (
                select
                     dateValue,
                     amount,
                     category
                from #temp
           ) x
            pivot
            (
                 sum(amount)
                for category in (' + @cols + ')
            ) p '
```

```
exec sp_executeSql @query
```

Прочитайте Динамический SQL Pivot онлайн: https://riptutorial.com/ru/sql-server/topic/10751/динамический-sql-pivot

глава 32: Динамическое маскирование данных

Examples

Адрес электронной почты маски с использованием маскирования динамических данных

Если у вас есть столбец электронной почты, вы можете замаскировать его с помощью маски email ():

```
ALTER TABLE  Company
ALTER COLUMN Email ADD MASKED WITH (FUNCTION = 'email()')
```

Когда пользователь пытается выбрать электронную почту из таблицы Company, он получит что-то вроде следующих значений:

mXXX@XXXX.com

zXXX@XXXX.com

rXXX@XXXX.com

Добавление частичной маски в столбец

Вы можете добавить парциальную маску в столбец, который будет показывать несколько символов от начала и конца строки и показать маску вместо символов посередине:

```
ALTER TABLE  Company
ALTER COLUMN Phone ADD MASKED WITH (FUNCTION = 'partial(5,"XXXXXXX",2)')
```

В параметрах частичной функции вы можете указать, сколько значений будет показано в начале, сколько будет отображаться значений с конца и каков будет шаблон, который показан в середине.

Когда пользователь пытается выбрать электронную почту из таблицы Company, он получит что-то вроде следующих значений:

(381) XXXXXXX39

(360) XXXXXXX01

(415) XXXXXXX05

Отображение случайного значения из диапазона с использованием random () mask

Случайная маска будет показывать номер rundom из указанного диапазона вместо фактического значения:

```
ALTER TABLE  Product
ALTER COLUMN Price ADD MASKED WITH (FUNCTION = 'random(100,200)')
```

Обратите внимание, что в некоторых случаях отображаемое значение может соответствовать фактическому значению в столбце (если случайно выбранный номер соответствует значению в ячейке).

Добавление маски по умолчанию в столбце

Если вы добавите маску по умолчанию в столбец, вместо фактического значения в инструкции SELECT будет отображаться маска:

```
ALTER TABLE  Company
ALTER COLUMN Postcode ADD MASKED WITH (FUNCTION = 'default()')
```

Контроль, который может видеть незамасленные данные

Вы можете предоставить привилегированным пользователям право видеть незамасленные значения, используя следующий оператор:

```
GRANT UNMASK TO MyUser
```

Если у какого-либо пользователя уже есть разрешение маскировки, вы можете отменить это разрешение:

```
REVOKE UNMASK TO MyUser
```

Прочитайте Динамическое маскирование данных онлайн: https://riptutorial.com/ru/sql-server/topic/7052/динамическое-маскирование-данных

глава 33: ДЛЯ JSON

Examples

ДЛЯ JSON PATH

Форматирует результаты запроса SELECT как текст JSON. Предложение JSON PATH добавляется после запроса:

```
SELECT top 3 object_id, name, type, principal_id FROM sys.objects
FOR JSON PATH
```

Имена столбцов будут использоваться как ключи в JSON, а значения ячеек будут генерироваться как значения JSON. Результатом запроса будет массив объектов JSON:

```
[
  {"object_id":3,"name":"sysrscols","type":"S "},
  {"object_id":5,"name":"sysrowsets","type":"S "},
  {"object_id":6,"name":"sysclones","type":"S "}
]
```

Значения NULL в столбце main_id будут игнорироваться (они не будут сгенерированы).

ДЛЯ JSON PATH с псевдонимами столбцов

Для JSON PATH вы можете управлять форматом вывода JSON с использованием псевдонимов столбцов:

```
SELECT top 3 object_id as id, name as [data.name], type as [data.type]
FROM sys.objects
FOR JSON PATH
```

В качестве имени ключа будет использоваться псевдоним столбца. Атрибуты столбцов с разделителями в точках (data.name и data.type) будут генерироваться как вложенные объекты. Если два столбца имеют один и тот же префикс в точечной нотации, они будут сгруппированы в один объект (данные в этом примере):

```
[
  {"id":3,"data":{"name":"sysrscols","type":"S "}},
  {"id":5,"data":{"name":"sysrowsets","type":"S "}},
  {"id":6,"data":{"name":"sysclones","type":"S "}}
]
```

Для предложения JSON без оболочки массива (отдельный объект на выходе)

Параметр WITHOUT_ARRAY_WRAPPER позволяет вам генерировать один объект вместо массива. Используйте этот параметр, если вы знаете, что вернете одну строку / объект:

```
SELECT top 3 object_id, name, type, principal_id
FROM sys.objects
WHERE object_id = 3
FOR JSON PATH, WITHOUT_ARRAY_WRAPPER
```

В этом случае возвращается один объект:

```
{"object_id":3,"name":"sysrscols","type":"S "}
```

INCLUDE_NULL_VALUES

Для предложения JSON игнорирует значения NULL в ячейках. Если вы хотите сгенерировать «ключ»: нулевые пары для ячеек, которые содержат значения NULL, добавьте параметр INCLUDE_NULL_VALUES в запрос:

```
SELECT top 3 object_id, name, type, principal_id
FROM sys.objects
FOR JSON PATH, INCLUDE_NULL_VALUES
```

Значения NULL в столбце main_id будут сгенерированы:

```
[
  {"object_id":3,"name":"sysrscols","type":"S ","principal_id":null},
  {"object_id":5,"name":"sysrowsets","type":"S ","principal_id":null},
  {"object_id":6,"name":"sysclones","type":"S ","principal_id":null}
]
```

Обрезка результатов с помощью объекта ROOT

Wraps возвращает массив JSON в дополнительный корневой объект с указанным ключом:

```
SELECT top 3 object_id, name, type FROM sys.objects
FOR JSON PATH, ROOT('data')
```

Результатом запроса будет массив объектов JSON внутри объекта-обертки:

```
{
  "data":[
            {"object_id":3,"name":"sysrscols","type":"S "},
            {"object_id":5,"name":"sysrowsets","type":"S "},
            {"object_id":6,"name":"sysclones","type":"S "}
          ]
}
```

ДЛЯ JSON AUTO

Автоматически вставляет значения из второй таблицы в виде вложенного подматрица объектов JSON:

```
SELECT top 5 o.object_id, o.name, c.column_id, c.name
FROM sys.objects o
    JOIN sys.columns c ON o.object_id = c.object_id
FOR JSON AUTO
```

Результатом запроса будет массив объектов JSON:

```
[
  {
   "object_id":3,
   "name":"sysrscols",
   "c":[
        {"column_id":12,"name":"bitpos"},
        {"column_id":6,"name":"cid"}
       ]
  },
  {
    "object_id":5,
    "name":"sysrowsets",
    "c":[
         {"column_id":13,"name":"colguid"},
         {"column_id":3,"name":"hbcolid"},
         {"column_id":8,"name":"maxinrowlen"}
     ]
  }
]
```

Создание пользовательской вложенной структуры JSON

Если вам нужна сложная структура JSON, которая не может быть создана с помощью FOR JSON PATH или FOR JSON AUTO, вы можете настроить свой вывод JSON, поставив подзапросы FOR JSON в виде выражений столбца:

```
SELECT top 5 o.object_id, o.name,
        (SELECT column_id, c.name
            FROM sys.columns c WHERE o.object_id = c.object_id
            FOR JSON PATH) as columns,
        (SELECT parameter_id, name
            FROM sys.parameters p WHERE o.object_id = p.object_id
            FOR JSON PATH) as parameters
FROM sys.objects o
FOR JSON PATH
```

Каждый подзапрос даст результат JSON, который будет включен в основной контент JSON.

Прочитайте ДЛЯ JSON онлайн: https://riptutorial.com/ru/sql-server/topic/4661/для-json

глава 34: ЕСЛИ ЕЩЕ

Examples

Единый оператор IF

Как и большинство других языков программирования, T-SQL также поддерживает инструкции IF..ELSE.

Например, в приведенном ниже примере `1 = 1` - выражение, которое оценивает значение True, и элемент управления входит в блок `BEGIN..END` а оператор Print печатает строку `'One is equal to One'`

```
IF ( 1 = 1)  --<-- Some Expression
 BEGIN
   PRINT 'One is equal to One'
 END
```

Несколько сообщений IF

Мы можем использовать несколько операторов IF для проверки нескольких выражений, полностью независимых друг от друга.

В приведенном ниже примере выражение каждого оператора `IF` оценивается, и если оно верно, выполняется код внутри блока `BEGIN...END` . В этом конкретном примере выражения First и Third являются истинными, и будут выполняться только эти операторы печати.

```
IF (1 = 1)  --<-- Some Expression      --<-- This is true
BEGIN
    PRINT 'First IF is True'           --<-- this will be executed
END

IF (1 = 2)  --<-- Some Expression
BEGIN
    PRINT 'Second IF is True'
END

IF (3 = 3)  --<-- Some Expression        --<-- This true
BEGIN
    PRINT 'Thrid IF is True'             --<-- this will be executed
END
```

Единый оператор IF..ELSE

В одном `IF..ELSE` , если выражение принимает значение True в выражении `IF` элемент управления входит в первый блок `BEGIN..END` и только код внутри этого блока выполняется, блок Else просто игнорируется.

С другой стороны, если выражение оценивается как `False` блок `ELSE BEGIN..END` выполняется, и элемент управления никогда не входит в первый блок `BEGIN..END` .

В приведенном ниже примере выражение будет оцениваться как false, а блок Else будет выполнен с печатью строки `'First expression was not true'`

```
IF ( 1 <> 1)  --<-- Some Expression
 BEGIN
     PRINT 'One is equal to One'
 END
ELSE
 BEGIN
     PRINT 'First expression was not true'
 END
```

Несколько IF ... ELSE с окончательными ELSE-сообщениями

Если у нас есть несколько операторов `IF...ELSE IF` но мы также хотим также выполнить какой-то фрагмент кода, если ни одно из выражений не оценивается как True, тогда мы можем просто добавить окончательный `ELSE` блок, который запускается только в том случае, если ни один из `IF` или `ELSE IF` выражены в true.

В приведенном ниже примере ни одно из выражений `IF` или `ELSE IF` является истинным, поэтому выполняется только блок `ELSE` и печатает `'No other expression is true'`

```
IF ( 1 = 1 + 1 )
    BEGIN
       PRINT 'First If Condition'
    END
ELSE IF (1 = 2)
    BEGIN
        PRINT 'Second If Else Block'
    END
ELSE IF (1 = 3)
    BEGIN
        PRINT 'Third If Else Block'
    END
ELSE
    BEGIN
        PRINT 'No other expression is true'  --<-- Only this statement will be printed
    END
```

Несколько IF ... ELSE

Чаще всего нам нужно проверять несколько выражений и принимать конкретные действия на основе этих выражений. Эта ситуация обрабатывается с использованием нескольких `IF...ELSE IF` операторов.

В этом примере все выражения оцениваются сверху вниз. Как только выражение оценивается как true, выполняется код внутри этого блока. Если выражение не оценивается как true, ничего не выполняется.

```
IF (1 = 1 + 1)
BEGIN
    PRINT 'First If Condition'
END
ELSE IF (1 = 2)
BEGIN
    PRINT 'Second If Else Block'
END
ELSE IF (1 = 3)
BEGIN
    PRINT 'Third If Else Block'
END
ELSE IF (1 = 1)       --<-- This is True
BEGIN
    PRINT 'Last Else Block'  --<-- Only this statement will be printed
END
```

Прочитайте ЕСЛИ ЕЩЕ онлайн: https://riptutorial.com/ru/sql-server/topic/5186/если-еще

глава 35: ЗАВЕРШЕНИЕ

параметры

параметр	подробности
РАЗДЕЛЕНИЕ	Поле (ы), которое следует за PARTITION BY, является тем, которое «группировка» будет основываться на

замечания

Предложение OVER определяет окна или подмножество строки в наборе результатов запроса. Функция окна может применяться для установки и вычисления значения для каждой строки в наборе. Предложение OVER может использоваться с:

- Функции ранжирования
- Совокупные функции

поэтому кто-то может вычислить агрегированные значения, такие как скользящие средние, совокупные совокупности, текущие итоговые значения или верхние результаты по каждой группе.

Очень абстрактно мы можем сказать, что OVER ведет себя как GROUP BY. Однако OVER применяется для поля / столбца, а не для всего запроса, как GROUP BY.

Примечание # 1: В SQL Server 2008 (R2) ORDER BY не может использоваться с функциями агрегатного окна (ссылка).

Examples

Использование функций агрегирования с OVER

Используя таблицу Cars , мы вычислим общую, максимальную, минимальную и среднюю сумму денег, которую каждый из них потратил и много раз (COUNT), она привезла автомобиль для ремонта.

Id CustomerId MechanicId Модель Статус Общая стоимость

```
SELECT CustomerId,
       SUM(TotalCost) OVER(PARTITION BY CustomerId) AS Total,
       AVG(TotalCost) OVER(PARTITION BY CustomerId) AS Avg,
       COUNT(TotalCost) OVER(PARTITION BY CustomerId) AS Count,
       MIN(TotalCost) OVER(PARTITION BY CustomerId) AS Min,
```

```
       MAX(TotalCost) OVER(PARTITION BY CustomerId) AS Max
  FROM CarsTable
 WHERE Status = 'READY'
```

Остерегайтесь, что использование OVER таким образом не приведет к агрегации возвращаемых строк. Вышеприведенный запрос вернет следующее:

Пользовательский ИД	Всего	в среднем	подсчитывать	Min	Максимум
1	430	215	2	200	230
1	430	215	2	200	230

Дублированная строка (строки) может быть не такой полезной для целей отчетности.

Если вы хотите просто агрегировать данные, вам будет лучше использовать предложение GROUP BY вместе с соответствующими агрегатными функциями. Например:

```
SELECT CustomerId,
       SUM(TotalCost) AS Total,
       AVG(TotalCost) AS Avg,
       COUNT(TotalCost) AS Count,
       MIN(TotalCost) AS Min,
       MAX(TotalCost)  AS Max
  FROM CarsTable
 WHERE Status = 'READY'
GROUP BY CustomerId
```

Суммарная сумма

Используя таблицу продаж товаров, мы постараемся выяснить, как продажи наших товаров увеличиваются через даты. Для этого мы рассчитаем *совокупную* сумму общих продаж по порядку заказа на дату продажи.

```
SELECT item_id, sale_Date
       SUM(quantity * price) OVER(PARTITION BY item_id ORDER BY sale_Date ROWS BETWEEN
UNBOUNDED PRECEDING) AS SalesTotal
  FROM SalesTable
```

Использование функций агрегирования для поиска последних записей

Используя библиотечную базу данных , мы пытаемся найти последнюю книгу, добавленную в базу данных для каждого автора. Для этого простого примера мы принимаем всегда увеличивающийся Id для каждой добавленной записи.

```
SELECT MostRecentBook.Name, MostRecentBook.Title
FROM ( SELECT Authors.Name,
              Books.Title,
              RANK() OVER (PARTITION BY Authors.Id ORDER BY Books.Id DESC) AS NewestRank
```

```
       FROM Authors
       JOIN Books ON Books.AuthorId = Authors.Id
     ) MostRecentBook
WHERE MostRecentBook.NewestRank = 1
```

Вместо RANK могут быть использованы две другие функции. В предыдущем примере результат будет таким же, но они дают разные результаты, когда упорядочение дает несколько строк для каждого ранга.

- `RANK()` : дубликаты получают одинаковый ранг, следующий ранг учитывает количество дубликатов предыдущего ранга
- `DENSE_RANK()` : дубликаты получают одинаковый ранг, следующий ранг всегда один выше предыдущего
- `ROW_NUMBER()` : даст каждой строке уникальный «ранг», «ранжирование» дубликатов случайным образом

Например, если в таблице был не уникальный столбец CreationDate, и заказ был выполнен на основе этого, следующий запрос:

```
SELECT Authors.Name,
       Books.Title,
       Books.CreationDate,
       RANK() OVER (PARTITION BY Authors.Id ORDER BY Books.CreationDate DESC) AS RANK,
       DENSE_RANK() OVER (PARTITION BY Authors.Id ORDER BY Books.CreationDate DESC) AS
DENSE_RANK,
       ROW_NUMBER() OVER (PARTITION BY Authors.Id ORDER BY Books.CreationDate DESC) AS
ROW_NUMBER,
FROM Authors
JOIN Books ON Books.AuthorId = Authors.Id
```

Может привести к:

автор	заглавие	Дата создания	РАНГ	DENSE_RANK	ROW_NUMBER
Автор 1	Книга 1	22/07/2016	1	1	1
Автор 1	Книга 2	22/07/2016	1	1	2
Автор 1	Книга 3	21/07/2016	3	2	3
Автор 1	Книга 4	21/07/2016	3	2	4
Автор 1	Книга 5	21/07/2016	3	2	5
Автор 1	Книга 6	04/07/2016	6	3	6
Автор 2	Книга 7	04/07/2016	1	1	1

Разделение данных на равноразделные ковши с использованием NTILE

Предположим, что у вас есть экзамены для нескольких экзаменов, и вы хотите разделить их на квартили на экзамен.

```
-- Setup data:
declare @values table(Id int identity(1,1) primary key, [Value] float, ExamId int)
insert into @values ([Value], ExamId) values
(65, 1), (40, 1), (99, 1), (100, 1), (90, 1), -- Exam 1 Scores
(91, 2), (88, 2), (83, 2), (91, 2), (78, 2), (67, 2), (77, 2) -- Exam 2 Scores

-- Separate into four buckets per exam:
select ExamId,
       ntile(4) over (partition by ExamId order by [Value] desc) as Quartile,
       Value, Id
from @values
order by ExamId, Quartile
```

	ExamId	Quartile	Value	Id
1	1	1	100	4
2	1	1	99	3
3	1	2	90	5
4	1	3	65	1
5	1	4	40	2
6	2	1	91	9
7	2	1	91	6
8	2	2	88	7
9	2	2	83	8
10	2	3	78	10
11	2	3	77	12
12	2	4	67	11

`ntile` работает, когда вам действительно нужно определенное количество ведер, и каждый заполняется примерно до одного уровня. Обратите внимание, что было бы тривиально отделить эти оценки от процентилей, просто используя `ntile(100)` .

Прочитайте ЗАВЕРШЕНИЕ онлайн: https://riptutorial.com/ru/sql-server/topic/353/завершение

глава 36: Запланированная задача или задание

Вступление

Агент SQL Server использует SQL Server для хранения информации о задании. Задания содержат один или несколько шагов работы. Каждый шаг содержит свою собственную задачу, то есть: резервное копирование базы данных. Агент SQL Server может запускать задание по расписанию, в ответ на конкретное событие или по требованию.

Examples

Создание запланированного задания

Создать работу

- Чтобы добавить работу сначала, мы должны использовать хранимую процедуру sp_add_job

```
USE msdb ;
GO
EXEC dbo.sp_add_job
@job_name = N'Weekly Job' ;  -- the job name
```

- Затем мы должны добавить шаг задания, используя хранимую процедуру с именем sp_add_jobStep

```
EXEC sp_add_jobstep
@job_name = N'Weekly Job',  -- Job name to add a step
@step_name = N'Set database to read only',  -- step name
@subsystem = N'TSQL',  -- Step type
@command = N'ALTER DATABASE SALES SET READ_ONLY',   -- Command
@retry_attempts = 5,  --Number of attempts
@retry_interval = 5 ; -- in minutes
```

- Направить задание на сервер

```
EXEC dbo.sp_add_jobserver
@job_name = N'Weekly Sales Data Backup',
@server_name = 'MyPC\data;   -- Default is LOCAL
GO
```

Создание расписания с использованием SQL

Чтобы создать расписание, мы должны использовать системную хранимую процедуру

sp_add_schedule

```
USE msdb
GO

EXEC sp_add_schedule
    @schedule_name = N'NightlyJobs' ,  -- specify the schedule name
    @freq_type = 4,   -- A value indicating when a job is to be executed (4) means Daily
    @freq_interval = 1,  -- The days that a job is executed and depends on the value of
`freq_type`.
    @active_start_time = 010000 ;   -- The time on which execution of a job can begin
GO
```

Есть больше параметров, которые можно использовать с `sp_add_schedule` вы можете узнать больше в приведенной выше ссылке.

Прикрепление графика к заданию

Чтобы приложить расписание к заданию агента SQL, вы должны использовать хранимую процедуру sp_attach_schedule

```
-- attaches the schedule to the job BackupDatabase
EXEC sp_attach_schedule
   @job_name = N'BackupDatabase',  -- The job name to attach with
   @schedule_name = N'NightlyJobs' ;  -- The schedule name
GO
```

Прочитайте Запланированная задача или задание онлайн: https://riptutorial.com/ru/sql-server/topic/5329/запланированная-задача-или-задание

глава 37: Запрос результатов по странице

Examples

ROW_NUMBER ()

```
SELECT Row_Number() OVER(ORDER BY UserName) As RowID, UserFirstName, UserLastName
FROM Users
```

Из которого он даст результат с полем RowID, которое вы можете использовать для перехода между страницами.

```
SELECT *
FROM
    ( SELECT Row_Number() OVER(ORDER BY UserName) As RowID, UserFirstName, UserLastName
      FROM Users
    ) As RowResults
WHERE RowID Between 5 AND 10
```

Прочитайте Запрос результатов по странице онлайн: https://riptutorial.com/ru/sql-server/topic/5803/запрос-результатов-по-странице

глава 38: Запросы с данными JSON

Examples

Использование значений из JSON в запросе

Функция JSON_VALUE позволяет вам извлекать данные из текста JSON по пути, указанному в качестве второго аргумента, и использовать это значение в любой части запроса выбора:

```
select ProductID, Name, Color, Size, Price, JSON_VALUE(Data, '$.Type') as Type
from Product
where JSON_VALUE(Data, '$.Type') = 'part'
```

Использование значений JSON в отчетах

Когда значения JSON извлекаются из текста JSON, вы можете использовать их в любой части запроса. Вы можете создать какой-то отчет по данным JSON с группировкой агрегатов и т. Д .:

```
select JSON_VALUE(Data, '$.Type') as type,
        AVG( cast(JSON_VALUE(Data, '$.ManufacturingCost') as float) ) as cost
from Product
group by JSON_VALUE(Data, '$.Type')
having JSON_VALUE(Data, '$.Type') is not null
```

Отфильтровать плохой текст JSON из результатов запроса

Если некоторый текст JSON может быть неправильно отформатирован, вы можете удалить эти записи из запроса с помощью функции ISJSON.

```
select ProductID, Name, Color, Size, Price, JSON_VALUE(Data, '$.Type') as Type
from Product
where JSON_VALUE(Data, '$.Type') = 'part'
and ISJSON(Data) > 0
```

Обновить значение в столбце JSON

Функция JSON_MODIFY может использоваться для обновления значения на каком-то пути. Вы можете использовать эту функцию для изменения исходного значения ячейки JSON в инструкции UPDATE:

```
update Product
set Data = JSON_MODIFY(Data, '$.Price', 24.99)
where ProductID = 17;
```

Функция JSON_MODIFY обновит или создаст ценовой ключ (если он не существует). Если новое значение равно NULL, ключ будет удален. Функция JSON_MODIFY будет обрабатывать новое значение в виде строки (специальные символы escape, обернуть ее двойными кавычками для создания правильной строки JSON). Если ваше новое значение - фрагмент JSON, вы должны обернуть его функцией JSON_QUERY:

```
update Product
set Data = JSON_MODIFY(Data, '$.tags', JSON_QUERY('["promo","new"]'))
where ProductID = 17;
```

Функция JSON_QUERY без второго параметра ведет себя как «приведение в JSON». Поскольку результатом JSON_QUERY является действительный фрагмент JSON (объект или массив), JSON_MODIFY не сможет избежать этого значения при изменении ввода JSON.

Добавить новое значение в массив JSON

Функция JSON_MODIFY может использоваться для добавления нового значения в некоторый массив внутри JSON:

```
update Product
set Data = JSON_MODIFY(Data, 'append $.tags', "sales")
where ProductID = 17;
```

Новое значение будет добавлено в конце массива, или будет создан новый массив со значением ["sales"]. Функция JSON_MODIFY будет обрабатывать новое значение в виде строки (специальные символы escape, обернуть ее двойными кавычками для создания правильной строки JSON). Если ваше новое значение - фрагмент JSON, вы должны обернуть его функцией JSON_QUERY:

```
update Product
set Data = JSON_MODIFY(Data, 'append $.tags', JSON_QUERY('{"type":"new"}'))
where ProductID = 17;
```

Функция JSON_QUERY без второго параметра ведет себя как «приведение в JSON». Поскольку результатом JSON_QUERY является действительный фрагмент JSON (объект или массив), JSON_MODIFY не сможет избежать этого значения при изменении ввода JSON.

Стол JOIN с внутренней коллекцией JSON

Если у вас есть «дочерняя таблица», отформатированная как коллекция JSON и сохраненная в строке как столбец JSON, вы можете распаковать эту коллекцию, преобразовать ее в таблицу и присоединиться к ней с родительской строкой. Вместо стандартного оператора JOIN вы должны использовать CROSS APPLY. В этом примере части продукта отформатированы как коллекция объектов JSON и хранятся в столбце

данных:

```
select ProductID, Name, Size, Price, Quantity, PartName, Code
from Product
    CROSS APPLY OPENJSON(Data, '$.Parts') WITH (PartName varchar(20), Code varchar(5))
```

Результат запроса эквивалентен соединению между таблицами Product и Part.

Поиск строк, содержащих значение в массиве JSON

В этом примере массив тегов может содержать различные ключевые слова, такие как [« promo», «sales»], поэтому мы можем открыть этот массив и значения фильтра:

```
select ProductID, Name, Color, Size, Price, Quantity
from Product
    CROSS APPLY OPENJSON(Data, '$.Tags')
where value = 'sales'
```

OPENJSON откроет внутреннюю коллекцию тегов и вернет ее в виде таблицы. Затем мы можем фильтровать результаты по некоторому значению в таблице.

Прочитайте Запросы с данными JSON онлайн: https://riptutorial.com/ru/sql-server/topic/5028/запросы-с-данными-json

глава 39: Заявление CASE

замечания

Вышеприведенный пример - просто показать синтаксис для использования операторов case в SQL Server с примером дня недели. Хотя такой же вывод можно получить, используя «SELECT DATENAME (WEEKDAY, GETDATE ())».

Examples

Простой оператор CASE

В простейшем случае одно значение или переменная проверяется на несколько возможных ответов. Приведенный ниже код является примером простого случая:

```
SELECT CASE DATEPART(WEEKDAY, GETDATE())
    WHEN 1 THEN 'Sunday'
    WHEN 2 THEN 'Monday'
    WHEN 3 THEN 'Tuesday'
    WHEN 4 THEN 'Wednesday'
    WHEN 5 THEN 'Thursday'
    WHEN 6 THEN 'Friday'
    WHEN 7 THEN 'Saturday'
END
```

Искаженный оператор CASE

В заявлении Searched Case каждый параметр может проверять одно или несколько значений независимо. Нижеприведенный код является примером поиска по делу:

```
DECLARE @FirstName varchar(30) = 'John'
DECLARE @LastName varchar(30) = 'Smith'

SELECT CASE
    WHEN LEFT(@FirstName, 1) IN ('a','e','i','o','u')
        THEN 'First name starts with a vowel'
    WHEN LEFT(@LastName, 1) IN ('a','e','i','o','u')
        THEN 'Last name starts with a vowel'
    ELSE
        'Neither name starts with a vowel'
END
```

Прочитайте Заявление CASE онлайн: https://riptutorial.com/ru/sql-server/topic/7238/заявление-case

глава 40: Значения NULL

Вступление

В SQL Server `NULL` представляет данные, которые отсутствуют или неизвестны. Это означает, что `NULL` - это не значение; он лучше описывается как заполнитель для значения. Это также причина, по которой вы не можете сравнивать `NULL` с любым значением, и даже с другим `NULL` .

замечания

SQL Server предоставляет другие методы для обработки нулей, таких как `IS NULL` , `IS NOT NULL` , `ISNULL()` , `COALESCE()` и другие.

Examples

Сравнение NULL

`NULL` - это особый случай, когда дело касается сравнений.

Предположим следующие данные.

```
id someVal
 ----
 0 NULL
 1 1
 2 2
```

С запросом:

```
 SELECT id
 FROM table
 WHERE someVal = 1
```

вернет id `1`

```
 SELECT id
 FROM table
 WHERE someVal <> 1
```

вернет id `2`

```
 SELECT id
 FROM table
 WHERE someVal IS NULL
```

вернет id `0`

```
SELECT id
FROM table
WHERE someVal IS NOT NULL
```

вернет оба идентификатора `1` и `2` .

Если вы хотите, чтобы NULL «подсчитывались» как значения в сравнении a `=` , `<>` , его сначала нужно преобразовать в счетный тип данных:

```
SELECT id
FROM table
WHERE ISNULL(someVal, -1) <> 1
```

ИЛИ ЖЕ

```
SELECT id
FROM table
WHERE someVal IS NULL OR someVal <> 1
```

возвращает `0` и `2`

Или вы можете изменить настройку ANSI Null .

ANSI NULLS

Из MSDN

> В будущей версии SQL Server ANSI_NULLS всегда будет включен, и любые приложения, которые явно устанавливают значение OFF, генерируют ошибку. Избегайте использования этой функции в новых разработках и планируйте изменять приложения, которые в настоящее время используют эту функцию.

`ANSI NULLS` , устанавливаемый в off, позволяет использовать a `=` / `<>` сравнение нулевых значений.

Учитывая следующие данные:

```
id someVal
 ----
 0 NULL
 1 1
 2 2
```

И с ANSI NULLS on этот запрос:

```
SELECT id
FROM table
```

```
WHERE someVal = NULL
```

не даст никаких результатов. Однако тот же запрос, когда ANSI NULLS выключен:

```
set ansi_nulls off

SELECT id
FROM table
WHERE someVal = NULL
```

Вернул бы id `0` .

НУЛЕВОЙ()

Функция `IsNull()` принимает два параметра и возвращает второй параметр, если первый имеет значение `null` .

Параметры:

1. проверьте выражение. Любое выражение любого типа данных.
2. восстановительная стоимость. Это значение, которое будет возвращено, если выражение проверки равно null. Значение замены должно быть типа данных, которое может быть неявно преобразовано в тип данных выражения проверки.

Функция `IsNull()` возвращает тот же тип данных, что и выражение проверки.

```
DECLARE @MyInt int -- All variables are null until they are set with values.

SELECT ISNULL(@MyInt, 3) -- Returns 3.
```

См. Также `COALESCE` , выше

Is null / Is not null

Поскольку значение null не является значением, вы не можете использовать операторы сравнения с нулями.
Чтобы проверить, имеет ли столбец или переменную значение null, вам нужно использовать значение `is null` :

```
DECLARE @Date date = '2016-08-03'
```

Следующий оператор выберет значение `6` , так как все сравнения с нулевыми значениями оцениваются как ложные или неизвестные:

```
SELECT CASE WHEN @Date = NULL THEN 1
            WHEN @Date <> NULL THEN 2
            WHEN @Date > NULL THEN 3
            WHEN @Date < NULL THEN 4
```

```
            WHEN @Date IS NULL THEN 5
            WHEN @Date IS NOT NULL THEN 6
```

Задав для содержимого переменной @Date значение `null` и повторите попытку, следующий оператор вернет `5` :

```
SET @Date = NULL -- Note that the '=' here is an assignment operator!

SELECT CASE WHEN @Date = NULL THEN 1
            WHEN @Date <> NULL THEN 2
            WHEN @Date > NULL THEN 3
            WHEN @Date < NULL THEN 4
            WHEN @Date IS NULL THEN 5
            WHEN @Date IS NOT NULL THEN 6
```

COALESCE ()

`COALESCE ()` Вычисляет аргументы в порядке и возвращает текущее значение первого выражения, которое изначально не оценивается в `NULL` .

```
DECLARE @MyInt int -- variable is null until it is set with value.
DECLARE @MyInt2 int -- variable is null until it is set with value.
DECLARE @MyInt3 int -- variable is null until it is set with value.

SET @MyInt3  = 3

SELECT COALESCE (@MyInt, @MyInt2 ,@MyInt3 ,5) -- Returns 3 : value of @MyInt3.
```

Хотя ISNULL () работает аналогично COALESCE (), функция ISNULL () принимает только два параметра - один для проверки и один для использования, если первый параметр равен NULL. См. Также `ISNULL` , ниже

NULL с NOT IN SubQuery

При обработке не в подзапросе с нулевым значением в подзапросе нам нужно устранить NULLS, чтобы получить ожидаемые результаты

```
create table #outertable (i int)
create table #innertable (i int)

insert into #outertable (i) values (1), (2),(3),(4), (5)
insert into #innertable (i) values (2), (3), (null)

select * from #outertable where i in (select i from #innertable)
--2
--3
--So far so good

select * from #outertable where i not in (select i from #innertable)
--Expectation here is to get 1,4,5 but it is not. It will get empty results because of the
NULL it executes as {select * from #outertable where i not in (null)}
```

```
--To fix this
select * from #outertable where i not in (select i from #innertable where i is not null)
--you will get expected results
--1
--4
--5
```

При обработке не в подзапросе с нулем будьте осторожны с ожидаемым результатом

Прочитайте Значения NULL онлайн: https://riptutorial.com/ru/sql-server/topic/5044/значения-null

глава 41: Изменить текст JSON

Examples

Изменить значение в тексте JSON по указанному пути

Функция JSON_MODIFY использует JSON-текст в качестве входного параметра и изменяет значение по указанному пути с использованием третьего аргумента:

```
declare @json nvarchar(4000) = N'{"Id":1,"Name":"Toy Car","Price":34.99}'
set @json = JSON_MODIFY(@json, '$.Price', 39.99)
print @json -- Output: {"Id":1,"Name":"Toy Car","Price":39.99}
```

В результате у нас будет новый текст JSON с «Price»: 39.99, а другое значение не будет изменено. Если объект по указанному пути не существует, JSON_MODIFY вставляет пару ключ: значение.

Чтобы удалить пару «ключ: значение», поместите NULL в качестве нового значения:

```
declare @json nvarchar(4000) = N'{"Id":1,"Name":"Toy Car","Price":34.99}'
set @json = JSON_MODIFY(@json, '$.Price', NULL)
print @json -- Output: {"Id":1,"Name":"Toy Car"}
```

JSON_MODIFY по умолчанию удаляет ключ, если он не имеет значения, поэтому вы можете использовать его для удаления ключа.

Добавить скалярное значение в массив JSON

JSON_MODIFY имеет режим «append», который добавляет значение в массив.

```
declare @json nvarchar(4000) = N'{"Id":1,"Name":"Toy Car","Tags":["toy","game"]}'
set @json = JSON_MODIFY(@json, 'append $.Tags', 'sales')
print @json -- Output: {"Id":1,"Name":"Toy Car","Tags":["toy","game","sales"]}
```

Если массив по указанному пути не существует, JSON_MODIFY (append) создаст новый массив с одним элементом:

```
declare @json nvarchar(4000) = N'{"Id":1,"Name":"Toy Car","Price":34.99}'
set @json = JSON_MODIFY(@json, 'append $.Tags', 'sales')
print @json -- Output {"Id":1,"Name":"Toy Car","Tags":["sales"]}
```

Вставить новый JSON-объект в текст JSON

Функция JSON_MODIFY позволяет вставлять объекты JSON в текст JSON:

```
declare @json nvarchar(4000) = N'{"Id":1,"Name":"Toy Car"}'
set @json = JSON_MODIFY(@json, '$.Price',
                        JSON_QUERY('{"Min":34.99,"Recommended":45.49}'))
print @json
-- Output: {"Id":1,"Name":"Toy Car","Price":{"Min":34.99,"Recommended":45.49}}
```

Поскольку третьим параметром является текст, вам необходимо обернуть его с помощью функции JSON_QUERY для «трансляции» текста в JSON. Без этого «приведения» JSON_MODIFY будет обрабатывать третий параметр как обычный текст и escape-символы, прежде чем вставлять его как строковое значение. Без результатов JSON_QUERY будут:

```
{"Id":1,"Name":"Toy Car","Price":'{\"Min\":34.99,\"Recommended\":45.49}'}
```

JSON_MODIFY вставляет этот объект, если он не существует, или удаляет его, если значение третьего параметра равно NULL.

Вставить новый массив JSON, сгенерированный с помощью запроса FOR JSON

Вы можете сгенерировать объект JSON с помощью стандартного запроса SELECT с предложением FOR JSON и вставить его в текст JSON в качестве третьего параметра:

```
declare @json nvarchar(4000) = N'{"Id":17,"Name":"WWI"}'
set @json = JSON_MODIFY(@json, '$.tables',
                        (select name from sys.tables FOR JSON PATH) )
print @json

(1 row(s) affected)
{"Id":1,"Name":"master","tables":[{"name":"Colors"},{"name":"Colors_Archive"},{"name":"OrderLines"},{"n
```

JSON_MODIFY будет знать, что запрос select с предложением FOR JSON генерирует действительный массив JSON, и он просто вставляет его в текст JSON.

> Вы можете использовать все опции FOR JSON в запросе SELECT, **кроме WITHOUT_ARRAY_WRAPPER** , который будет генерировать один объект вместо массива JSON. См. Другой пример в этом разделе, чтобы увидеть, как вставить один объект JSON.

Вставить один объект JSON, сгенерированный с помощью предложения FOR JSON

Вы можете сгенерировать объект JSON с помощью стандартного запроса SELECT с предложением FOR JSON и WITHOUT_ARRAY_WRAPPER и вставить его в текст JSON в качестве третьего параметра:

```
declare @json nvarchar(4000) = N'{"Id":17,"Name":"WWI"}'
```

```
set @json = JSON_MODIFY(@json, '$.table',
                        JSON_QUERY(
                         (select name, create_date, schema_id
                           from sys.tables
                           where name = 'Colors'
                           FOR JSON PATH, WITHOUT_ARRAY_WRAPPER)))
print @json

(1 row(s) affected)
{"Id":17,"Name":"WWI","table":{"name":"Colors","create_date":"2016-06-
02T10:04:03.280","schema_id":13}}
```

Для JSON с параметром WITHOUT_ARRAY_WRAPPER может генерироваться недопустимый текст JSON, если запрос SELECT возвращает более одного результата (в этом случае вы должны использовать TOP 1 или фильтр по первичному ключу). Поэтому JSON_MODIFY предположит, что возвращаемый результат является просто обычным текстом и избегает его, как и любой другой текст, если вы не обертываете его функцией JSON_QUERY.

> Вы должны перенести **FOR JSON, WITHOUT_ARRAY_WRAPPER** запрос с **помощью** функции **JSON_QUERY** , чтобы **придать** результат JSON.

Прочитайте Изменить текст JSON онлайн: https://riptutorial.com/ru/sql-server/topic/6883/изменить-текст-json

глава 42: Имена псевдонимов на сервере Sql

Вступление

Вот несколько способов предоставления имен псевдонимов столбцам на Sql Server

Examples

Использование AS

Это метод ANSI SQL работает во всех РСУБД. Широко используемый подход.

```
CREATE TABLE AliasNameDemo (id INT,firstname VARCHAR(20),lastname VARCHAR(20))

INSERT INTO AliasNameDemo
VALUES      (1,'MyFirstName','MyLastName')

SELECT FirstName +' '+ LastName As FullName
FROM   AliasNameDemo
```

Использование =

Это мой предпочтительный подход. Ничто не связано с производительностью - это просто личный выбор. Это заставляет код выглядеть чистым. Вы можете легко увидеть результирующие имена столбцов вместо прокрутки кода, если у вас есть большое выражение.

```
CREATE TABLE AliasNameDemo (id INT,firstname VARCHAR(20),lastname VARCHAR(20))

INSERT INTO AliasNameDemo
VALUES      (1,'MyFirstName','MyLastName')

SELECT FullName = FirstName +' '+ LastName
FROM   AliasNameDemo
```

Предоставление псевдонима после имени производной таблицы

Это странный подход, который большинство людей не знает об этом.

```
CREATE TABLE AliasNameDemo(id INT,firstname VARCHAR(20),lastname VARCHAR(20))

INSERT INTO AliasNameDemo
VALUES      (1,'MyFirstName','MyLastName')

SELECT *
```

```
FROM   (SELECT firstname + ' ' + lastname
        FROM   AliasNameDemo) a (fullname)
```

- **демонстрация**

Без использования AS

Этот синтаксис будет похож на использование ключевого слова `AS` . Просто нам не нужно использовать ключевое слово `AS`

```
CREATE TABLE AliasNameDemo (id INT,firstname VARCHAR(20),lastname VARCHAR(20))

INSERT INTO AliasNameDemo
VALUES      (1,'MyFirstName','MyLastName')

SELECT FirstName +' '+ LastName FullName
FROM   AliasNameDemo
```

Прочитайте Имена псевдонимов на сервере Sql онлайн: https://riptutorial.com/ru/sql-server/topic/10784/имена-псевдонимов-на-сервере-sql

глава 43: Импорт BULK

Examples

BULK INSERT с опциями

Вы можете настроить правила синтаксического анализа, используя различные параметры в предложении WITH:

```
BULK INSERT People
FROM 'f:\orders\people.csv'
WITH  (  CODEPAGE = '65001',
         FIELDTERMINATOR =',',
         ROWTERMINATOR ='\n'
      );
```

В этом примере CODEPAGE указывает, что исходный файл в файле UTF-8 и TERMINATORS являются комой и новой строкой.

BULK INSERT

Команда BULK INSERT может использоваться для импорта файла в SQL Server:

```
BULK INSERT People
FROM 'f:\orders\people.csv'
```

Команда BULK INSERT будет отображать столбцы в файлах со столбцами в целевой таблице.

Чтение всего содержимого файла с помощью OPENROWSET (BULK)

Вы можете прочитать содержимое файла с помощью функции OPENROWSET (BULK) и сохранить содержимое в некоторой таблице:

```
INSERT INTO myTable(content)
   SELECT BulkColumn
          FROM OPENROWSET(BULK N'C:\Text1.txt', SINGLE_BLOB) AS Document;
```

Параметр SINGLE_BLOB будет считывать весь контент из файла в виде отдельной ячейки.

Чтение файла с использованием OPENROWSET (BULK) и файла формата

Yu может определить формат файла, который будет импортирован с помощью опции FORMATFILE:

```
INSERT INTO mytable
SELECT a.*
FROM OPENROWSET(BULK 'c:\test\values.txt',
   FORMATFILE = 'c:\test\values.fmt') AS a;
```

Файл формата format_file.fmt описывает столбцы в values.txt:

```
9.0
2
1  SQLCHAR  0  10 "\t"        1  ID                SQL_Latin1_General_Cp437_BIN
2  SQLCHAR  0  40 "\r\n"      2  Description       SQL_Latin1_General_Cp437_BIN
```

Чтение файла json с использованием OPENROWSET (BULK)

Вы можете использовать OPENROWSET для чтения содержимого файла и передать его другой функции, которая будет анализировать результаты.

В следующем примере показано горячее чтение всего содержимого файла JSON с помощью OPENROWSET (BULK), а затем предоставление функции BulkColumn функции OPENJSON, которая будет анализировать JSON и возвращать столбцы:

```
SELECT book.*
 FROM OPENROWSET (BULK 'C:\JSON\Books\books.json', SINGLE_CLOB) as j
 CROSS APPLY OPENJSON(BulkColumn)
       WITH( id nvarchar(100), name nvarchar(100), price float,
             pages int, author nvarchar(100)) AS book
```

Прочитайте Импорт BULK онлайн: https://riptutorial.com/ru/sql-server/topic/7330/импорт-bulk

глава 44: Индекс

Examples

Создать кластеризованный индекс

С кластеризованным индексом страницы листа содержат фактические строки таблицы. Следовательно, может быть только один кластеризованный индекс.

```
CREATE TABLE Employees
(
    ID CHAR(900),
    FirstName NVARCHAR(3000),
    LastName NVARCHAR(3000),
    StartYear CHAR(900)
)
GO

CREATE CLUSTERED INDEX IX_Clustered
ON Employees(ID)
GO
```

Создать некластерный индекс

Некластеризованные индексы имеют структуру, отдельную от строк данных. Некластеризованный индекс содержит некластеризованные значения ключа ключа, и каждая запись ключа имеет указатель на строку данных, содержащую значение ключа. На SQL Server 2008/2012 может быть максимум 999 некластеризованных индексов.

Ссылка для справки: https://msdn.microsoft.com/en-us/library/ms143432.aspx

```
CREATE TABLE Employees
(
    ID CHAR(900),
    FirstName NVARCHAR(3000),
    LastName NVARCHAR(3000),
    StartYear CHAR(900)
)
GO

CREATE NONCLUSTERED INDEX IX_NonClustered
ON Employees(StartYear)
GO
```

Показать информацию об индексе

```
SP_HELPINDEX tableName
```

Индекс на вид

```
CREATE VIEW  View_Index02
WITH SCHEMABINDING
AS
SELECT c.CompanyName, o.OrderDate, o.OrderID, od.ProductID
     FROM dbo.Customers C
        INNER JOIN dbo.orders O ON c.CustomerID=o.CustomerID
            INNER JOIN dbo.[Order Details] od ON o.OrderID=od.OrderID
GO

CREATE UNIQUE CLUSTERED INDEX IX1 ON
    View_Index02(OrderID, ProductID)
```

Индекс падения

```
DROP INDEX IX_NonClustered ON Employees
```

Возвращает индексы размера и фрагментации

```
sys.dm_db_index_physical_stats (
    { database_id | NULL | 0 | DEFAULT }
  , { object_id | NULL | 0 | DEFAULT }
  , { index_id | NULL | 0 | -1 | DEFAULT }
  , { partition_number | NULL | 0 | DEFAULT }
  , { mode | NULL | DEFAULT }
)


Sample :

SELECT * FROM sys.dm_db_index_physical_stats
    (DB_ID(N'DBName'), OBJECT_ID(N'IX_NonClustered '), NULL, NULL , 'DETAILED');
```

Реорганизация и восстановление индекса

Значение avg_fragmentation_in_percent	Корректирующее заявление
> 5% и <= 30%	REORGANIZE
> 30%	ВОССТАНОВЛЕНИЕ

```
ALTER INDEX IX_NonClustered ON tableName REORGANIZE;


ALTER INDEX ALL ON Production.Product
 REBUILD WITH (FILLFACTOR = 80, SORT_IN_TEMPDB = ON,
          STATISTICS_NORECOMPUTE = ON);
```

Перестроить или реорганизовать все индексы на таблицу

Восстановление индексов производится с использованием следующего оператора

```
ALTER INDEX All ON tableName REBUILD;
```

Это уменьшает индекс и воссоздает его, устраняет фрагментацию, восстанавливает дисковое пространство и индексирует страницы переупорядочения.

Можно также реорганизовать индекс, используя

```
ALTER INDEX All ON tableName REORGANIZE;
```

который будет использовать минимальные системные ресурсы и дефрагментирует листовой уровень кластеризованных и некластеризованных индексов на таблицах и представлениях, физически переупорядочивая страницы листового уровня, чтобы соответствовать логическому, слева направо, порядку листовых узлов

Перестроить всю базу данных индексов

```
EXEC sp_MSForEachTable 'ALTER INDEX ALL ON ? REBUILD'
```

Индексные исследования

Вы можете использовать «SP_HELPINDEX Table_Name», но у Kimberly Tripp есть хранимая процедура (которая может быть найдена здесь), что является лучшим примером, поскольку он показывает больше об индексах, включая столбцы и определение фильтра, например:
Использование:

```
USE Adventureworks
EXEC sp_SQLskills_SQL2012_helpindex 'dbo.Product'
```

В качестве альтернативы, Тибор Караси имеет хранимую процедуру (найденную здесь). Более поздняя версия также покажет информацию об использовании индекса, а также предоставит список предложений по индексу. Использование:

```
USE Adventureworks
EXEC sp_indexinfo 'dbo.Product'
```

Прочитайте Индекс онлайн: https://riptutorial.com/ru/sql-server/topic/4998/индекс

глава 45: Иностранные ключи

Examples

Внешние отношения / ограничение

Внешние ключи позволяют определить взаимосвязь между двумя таблицами. Одна (родительская) таблица должна иметь первичный ключ, который однозначно идентифицирует строки в таблице. Другая (дочерняя) таблица может иметь значение первичного ключа от родителя в одном из столбцов. Ограничение FOREIGN KEY REFERENCES гарантирует, что значения в дочерней таблице должны существовать в качестве значения первичного ключа в родительской таблице.

В этом примере у нас есть родительская таблица компании с основным ключом CompanyId и таблица Child Employee, в которой есть идентификатор компании, в которой работает этот сотрудник.

```
create table Company (
   CompanyId int primary key,
   Name nvarchar(200)
)
create table Employee (
    EmployeeId int,
    Name nvarchar(200),
    CompanyId int
        foreign key references Company(companyId)
)
```

ссылки на внешние ключи гарантируют, что значения, вставленные в столбец Employee.CompanyId, также должны существовать в столбце Company.CompanyId. Кроме того, никто не может удалять компанию в таблице компании, если есть хотя бы один сотрудник с подходящим companyId в дочерней таблице.

Отношение FOREIGN KEY гарантирует, что строки в двух таблицах не могут быть «несвязаны».

Поддержание отношений между родительскими / дочерними строками

Предположим, что у нас есть одна строка в таблице Company with companyId 1. Мы можем вставить строку в таблицу employee, в которой есть companyId 1:

```
insert into Employee values (17, 'John', 1)
```

Однако мы не можем вставить сотрудника, у которого есть несуществующий CompanyId:

```
insert into Employee values (17, 'John', 111111)
```

Msg 547, уровень 16, состояние 0, строка 12 Оператор INSERT противоречил ограничению FOREIGN KEY «FK__Employee__Compan__1EE485AA». Конфликт произошел в базе данных «MyDb», в таблице «dbo.Company», в столбце «CompanyId». Заявление было прекращено.

Кроме того, мы не можем удалить родительскую строку в таблице компании, если в таблице employee есть хотя бы одна дочерняя строка, которая ссылается на нее.

```
delete from company where CompanyId = 1
```

Msg 547, уровень 16, состояние 0, строка 14 Оператор DELETE противоречил ограничению REFERENCE «FK__Employee__Compan__1EE485AA». Конфликт произошел в базе данных «MyDb», таблице «dbo.Employee», в столбце «CompanyId». Заявление было прекращено.

Внешнее ключевое отношение гарантирует, что строки компании и сотрудника не будут «несвязаны».

Добавление отношения внешнего ключа к существующей таблице

Ограничение **FOREIGN KEY** может быть добавлено в существующие таблицы, которые все еще не связаны. Представьте, что у нас есть таблицы Company and Employee, где столбец Employee table CompanyId, но не имеет отношения с внешним ключом. Оператор ALTER TABLE позволяет добавить ограничение **внешнего ключа** в существующий столбец, который ссылается на другую таблицу и первичный ключ в этой таблице:

```
alter table Employee
    add  foreign key (CompanyId) references Company(CompanyId)
```

Добавить внешний ключ в существующую таблицу

Столбцы **FOREIGN KEY** с ограничением могут быть добавлены в существующие таблицы, которые все еще не находятся в отношениях. Представьте, что у нас есть таблицы компаний и сотрудников, в которых таблица Employee не имеет столбца CompanyId. Оператор ALTER TABLE позволяет добавить новый столбец с ограничениями **внешнего ключа,** который ссылается на другую таблицу и первичный ключ в этой таблице:

```
alter table Employee
    add CompanyId int foreign key references Company(CompanyId)
```

Получение информации о внешних ограничениях ключа

Системный вид sys.foreignkeys возвращает информацию обо всех отношениях внешнего ключа в базе данных:

```
select name,
 OBJECT_NAME(referenced_object_id) as [parent table],
 OBJECT_NAME(parent_object_id) as [child table],
 delete_referential_action_desc,
 update_referential_action_desc
from sys.foreign_keys
```

Прочитайте Иностранные ключи онлайн: https://riptutorial.com/ru/sql-server/topic/5355/иностранные-ключи

глава 46: Интеграция с обычным языком

Examples

Включение CLR в базе данных

По умолчанию CLR-процедуры не включены. Для включения CLR необходимо выполнить следующие запросы:

```
sp_configure 'show advanced options', 1;
GO
RECONFIGURE;
GO
sp_configure 'clr enabled', 1;
GO
RECONFIGURE;
GO
```

Кроме того, если какой-либо модуль CLR нуждается в внешнем доступе, вы должны установить для свойства TRUSTWORTHY значение ON в своей базе данных:

```
ALTER DATABASE MyDbWithClr SET TRUSTWORTHY ON
```

Добавление .dll, содержащего модули Sql CLR

Процедуры, функции, триггеры и типы, написанные на .Net-языках, хранятся в файлах DLL. Создав файл DLL, содержащий процедуры CLR, вы должны импортировать его в SQL Server:

```
CREATE ASSEMBLY MyLibrary
FROM 'C:\lib\MyStoredProcedures.dll'
    WITH PERMISSION_SET = EXTERNAL_ACCESS
```

PERMISSION_SET является безопасным по умолчанию, что означает, что для кода в .dll не требуется разрешение для доступа к внешним ресурсам (например, файлы, веб-сайты и другие серверы) и что он не будет использовать собственный код, который может получить доступ к памяти.

PERMISSION_SET = EXTERNAL_ACCESS используется для обозначения сборок, содержащих код, который будет обращаться к внешним ресурсам.

вы можете найти информацию о текущих файлах сборки CLR в представлении sys.assemblies:

```
SELECT *
FROM sys.assemblies asms
```

```
WHERE is_user_defined = 1
```

Создание функции CLR в SQL Server

Если вы создали функцию .Net, скомпилировали ее в DLL и импортировали на сервер SQL в качестве сборки, вы можете создать определенную пользователем функцию, которая ссылается на эту функцию:

```
CREATE FUNCTION dbo.TextCompress(@input nvarchar(max))
RETURNS varbinary(max)
AS EXTERNAL NAME MyLibrary.[Name.Space.ClassName].TextCompress
```

Вам нужно указать имя функции и подпись с входными параметрами и вернуть значения, соответствующие функции .Net. В предложении AS EXTERNAL NAME вам нужно указать имя сборки, пространство имен / имя класса, в котором находится эта функция, и имя метода в классе, который содержит код, который будет отображаться как функция.

Вы можете найти информацию о функциях CLR, используя следующий запрос:

```
SELECT * FROM dbo.sysobjects WHERE TYPE ='FS'
```

Создать CLR Пользовательский тип в SQL Server

Если вы создали класс .Net, который представляет определенный пользовательский тип, скомпилировал его в .dll и импортировал на сервер SQL в качестве сборки, вы можете создать пользовательскую функцию, которая ссылается на этот класс:

```
CREATE TYPE dbo.Point
EXTERNAL NAME MyLibrary.[Name.Space.Point]
```

Вам нужно указать имя типа, который будет использоваться в запросах T-SQL. В разделе EXTERNAL NAME вам нужно указать имя сборки, пространство имен и имя класса.

Создание процедуры CLR в SQL Server

Если вы создали метод .NET в каком-то классе, скомпилировали его в .dll и импортировали на сервер SQL в качестве сборки, вы можете создать пользовательскую хранимую процедуру, ссылающуюся на метод в этой сборке:

```
CREATE PROCEDURE dbo.DoSomethng(@input nvarchar(max))
AS EXTERNAL NAME MyLibrary.[Name.Space.ClassName].DoSomething
```

Вам нужно указать имя процедуры и подписи с входными параметрами, которые соответствуют методу .Net. В предложении AS EXTERNAL NAME вам нужно указать имя сборки, пространство имен / имя класса, в котором находится эта процедура, и имя метода

в классе, который содержит код, который будет отображаться как процедура.

Прочитайте Интеграция с обычным языком онлайн: https://riptutorial.com/ru/sql-server/topic/7116/интеграция-с-обычным-языком

глава 47: Использование таблицы TEMP

замечания

Временные таблицы действительно очень полезны.

Таблица может быть создана во время выполнения и может выполнять все операции, выполняемые в обычной таблице.

Эти таблицы создаются в базе данных tempdb.

Используется когда?

1. Мы должны выполнить сложную операцию соединения.
2. Мы выполняем большое количество манипуляций с строками в хранимых процедурах.
3. Может заменить использование курсора.

Таким образом, увеличивается производительность.

Examples

Локальная таблица темпов

- Будет доступно до тех пор, пока текущее соединение не будет сохранено для пользователя.

  Автоматически удаляется, когда пользователь отключается.

  Имя должно начинаться с # (#temp)

```
CREATE TABLE #LocalTempTable(
                StudentID      int,
                StudentName    varchar(50),
                StudentAddress varchar(150))
```

```
insert into #LocalTempTable values ( 1, 'Ram','India');

select * from #LocalTempTable
```

После выполнения всех этих операторов, если мы закроем окно запроса и снова откроем его и попробуем вставить и выберите его, появится сообщение об ошибке

```
“Invalid object name #LocalTempTable”
```

Глобальная таблица темпов

- Начнется с ## (## temp).

  Будет удалено, только если пользователь отключит все подключения.

  Он ведет себя как постоянный стол.

```
CREATE TABLE ##NewGlobalTempTable(
               StudentID      int,
               StudentName    varchar(50),
               StudentAddress varchar(150))

Insert Into ##NewGlobalTempTable values ( 1,'Ram','India');
Select * from ##NewGlobalTempTable
```

Примечание. Они доступны для просмотра всем пользователям базы данных, независимо от уровня разрешений.

Удаление временных таблиц

Таблицы Temp должны иметь уникальные идентификаторы (внутри сеанса, для локальных временных таблиц или внутри сервера для глобальных временных таблиц). При попытке создать таблицу с использованием уже существующего имени будет возвращена следующая ошибка:

```
There is already an object named '#tempTable' in the database.
```

Если ваш запрос создает временные таблицы, и вы хотите запускать его более одного раза, вам нужно будет отбросить таблицы, прежде чем пытаться их сгенерировать. Основной синтаксис для этого:

```
drop table #tempTable
```

Попытка выполнить этот синтаксис перед существованием таблицы (например, при первом запуске синтаксиса) вызовет еще одну ошибку:

```
Cannot drop the table '#tempTable', because it does not exist or you do not have permission.
```

Чтобы этого избежать, вы можете проверить, существует ли таблица, прежде чем отбрасывать ее, например:

```
IF OBJECT_ID ('tempdb..#tempTable', 'U') is not null DROP TABLE #tempTable
```

Прочитайте Использование таблицы TEMP онлайн: https://riptutorial.com/ru/sql-server/topic/5328/использование-таблицы-temp

глава 48: Исходно скомпилированные модули (Hekaton)

Examples

Собственно скомпилированная хранимая процедура

В процедуре с собственной компиляцией код T-SQL скомпилируется в dll и выполняется как собственный C-код. Чтобы создать хранимую процедуру Native Compiled, вам необходимо:

- Использовать стандартный синтаксис CREATE PROCEDURE
- Установите опцию NATIVE_COMPILATION в определении хранимой процедуры
- Используйте опцию SCHEMABINDING в определении хранимой процедуры
- Определить опцию EXECUTE AS OWNER в определении хранимой процедуры

Вместо стандартного блока BEGIN END вам необходимо использовать блок BEGIN ATOMIC:

```
BEGIN ATOMIC
   WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
   -- T-Sql code goes here
END
```

Пример:

```
CREATE PROCEDURE usp_LoadMemOptTable (@maxRows INT, @FullName NVARCHAR(200))
WITH
    NATIVE_COMPILATION,
    SCHEMABINDING,
    EXECUTE AS OWNER
AS
BEGIN ATOMIC
WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
    DECLARE @i INT = 1
    WHILE @i <= @maxRows
    BEGIN
        INSERT INTO dbo.MemOptTable3 VALUES(@i, @FullName, GETDATE())
        SET @i = @i+1
    END
END
GO
```

Настраиваемая скалярная функция

Код в изначально скомпилированной функции будет преобразован в C-код и скомпилирован как dll. Чтобы создать скалярную функцию Native Compiled, вам необходимо:

- Использовать стандартный синтаксис CREATE FUNCTION
- Установите опцию NATIVE_COMPILATION в определении функции
- Использовать параметр SCHEMABINDING в определении функции

Вместо стандартного блока BEGIN END вам необходимо использовать блок BEGIN ATOMIC:

```
BEGIN ATOMIC
   WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
   -- T-Sql code goes here
END
```

Пример:

```
CREATE FUNCTION [dbo].[udfMultiply]( @v1 int, @v2 int )
RETURNS bigint
WITH NATIVE_COMPILATION, SCHEMABINDING
AS
BEGIN ATOMIC WITH (TRANSACTION ISOLATION LEVEL = SNAPSHOT, LANGUAGE = N'English')

    DECLARE @ReturnValue bigint;
    SET @ReturnValue = @v1 * @v2;

    RETURN (@ReturnValue);
END

-- usage sample:
SELECT dbo.udfMultiply(10, 12)
```

Встроенная функция значения таблицы

Встроенная функция значения таблицы возвращает таблицу в качестве результата. Код в изначально скомпилированной функции будет преобразован в C-код и скомпилирован как dll. Только встроенные функции, поддерживающие таблицу, поддерживаются в версии 2016. Для создания собственной функции значения таблицы вам необходимо:

- Использовать стандартный синтаксис CREATE FUNCTION
- Установите опцию NATIVE_COMPILATION в определении функции
- Использовать параметр SCHEMABINDING в определении функции

Вместо стандартного блока BEGIN END вам необходимо использовать блок BEGIN ATOMIC:

```
BEGIN ATOMIC
   WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
   -- T-Sql code goes here
END
```

Пример:

```
CREATE FUNCTION [dbo].[udft_NativeGetBusinessDoc]
(
   @RunDate VARCHAR(25)
)
RETURNS TABLE
WITH SCHEMABINDING,
     NATIVE_COMPILATION
AS
     RETURN
(
    SELECT BusinessDocNo,
           ProductCode,
           UnitID,
           ReasonID,
           PriceID,
           RunDate,
           ReturnPercent,
           Qty,
           RewardAmount,
           ModifyDate,
           UserID
    FROM dbo.[BusinessDocDetail_11]
    WHERE RunDate >= @RunDate
);
```

Прочитайте Исходно скомпилированные модули (Hekaton) онлайн:
https://riptutorial.com/ru/sql-server/topic/6089/исходно-скомпилированные-модули--hekaton-

глава 49: Клавиши быстрого доступа Microsoft SQL Server Management Studio

Examples

Примеры ярлыков

1. Откройте новое окно запросов с текущим соединением (`Ctrl` + `N`)
2. Переключение между открытыми вкладками (`Ctrl` + `Tab`)
3. Показать / скрыть панель результатов (`Ctrl` + `R`)
4. Выполнить выделенный запрос (`Ctrl` + `E`)
5. Сделайте выделенный текст заглавным или строчным (`Ctrl` + `Shift` + `U` , `Ctrl` + `Shift` + `L`)
6. Член списка Intellisense и полное слово (`Ctrl` + `Space` , `Tab`)
7. Перейдите к строке (`Ctrl` + `G`)
8. закрыть вкладку в SQL Server Managament Studio (`Ctrl` + `F4`)

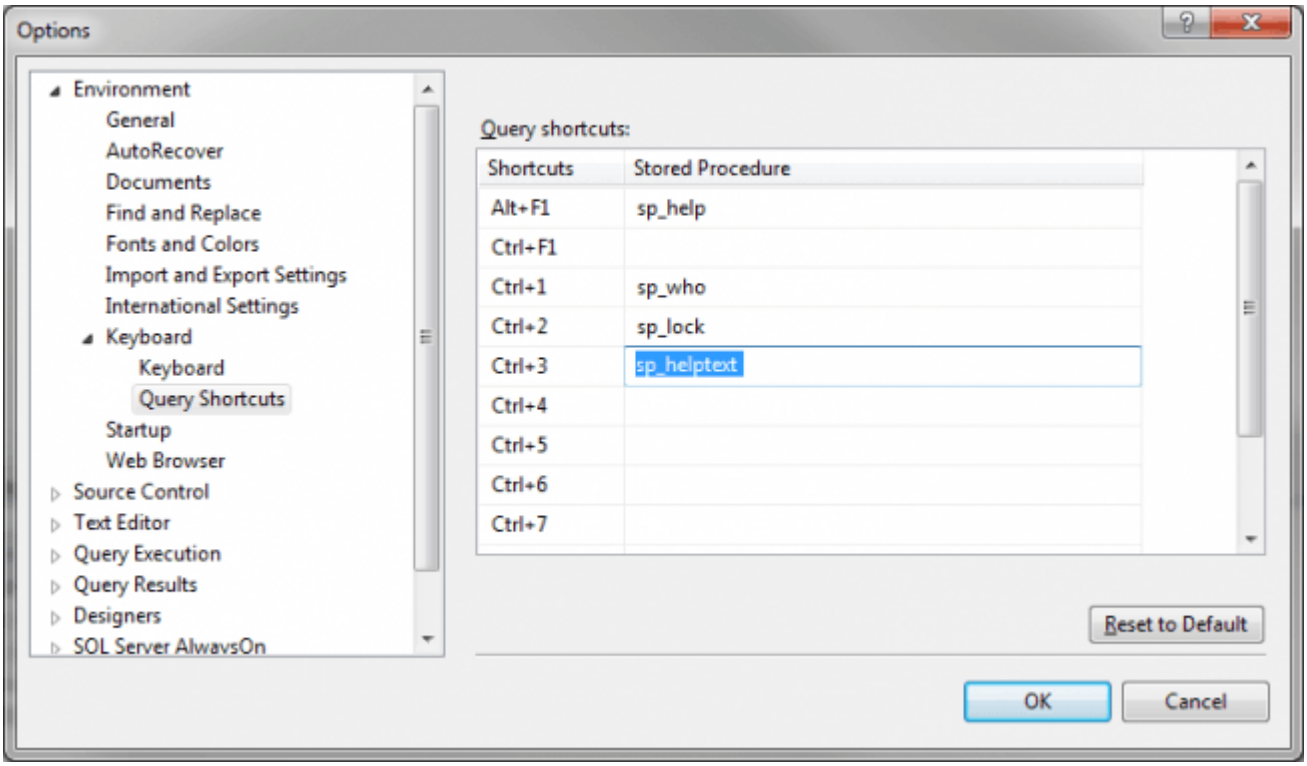
Клавиши быстрого доступа к меню

1. Перейдите в панель меню SQL Server Management Studio (`ALT`)
2. Активируйте меню для компонента инструмента (`ALT` + `HYPHEN`)
3. Отображение контекстного меню (`SHIFT` + `F`)
4. Отобразите диалоговое окно «Новый файл», чтобы создать файл (`CTRL` + `N`)
5. Отобразите диалоговое окно «Открыть проект», чтобы открыть существующий проект (`CTRL` + `SHIFT` + `O`)
6. Отобразите диалоговое окно «Добавить новый элемент», чтобы добавить новый файл в текущий проект (`CTRL` + `SHIFT` + `A`)
7. Отобразите диалоговое окно «Добавить существующий элемент», чтобы добавить существующий файл в текущий проект (`CTRL` + `SHIFT` + `A`)
8. Отобразить конструктор запросов (`CTRL` + `SHIFT` + `Q`)
9. Закройте меню или диалоговое окно, отменив действие (`ESC`)

Пользовательские сочетания клавиш

Откройте Инструменты -> Параметры. Перейти к среде -> Клавиатура -> Быстрый поиск

С правой стороны вы можете увидеть некоторые ярлыки, которые по умолчанию используются в SSMS. Теперь, если вам нужно добавить новый, просто нажмите на любой столбец в столбце «Сохраненная процедура».



Нажмите «OK». Теперь перейдите в окно запроса и выберите хранимую процедуру, затем нажмите CTRL + 3, он покажет результат хранимой процедуры.

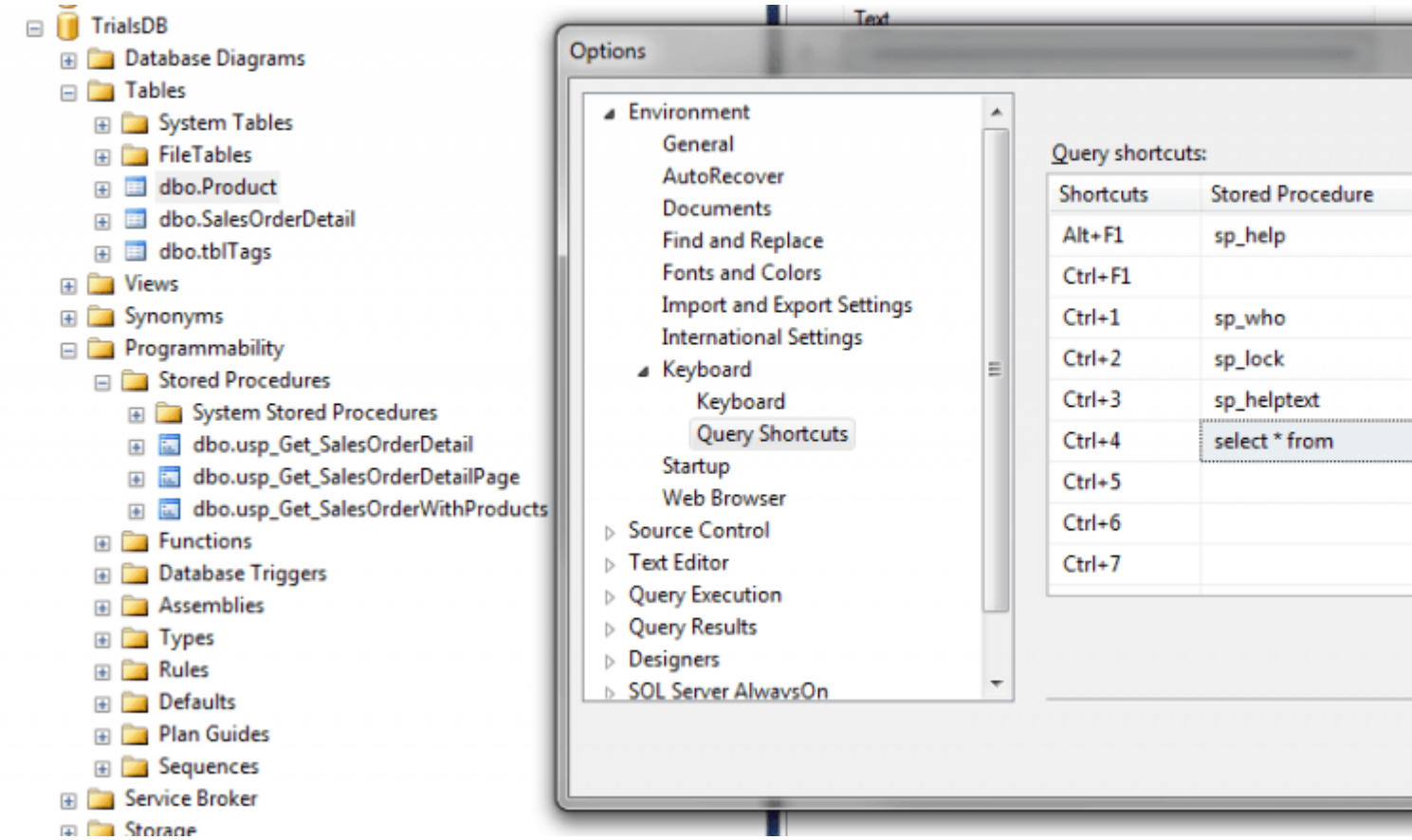
```
usp_Get_SalesOrderDetail
```

100 %

Results | Messages

	Text
1	-- ==
2	-- Author: <Author,Sibeesh Venu>
3	-- Create date: <Create Date, 18-Feb-2016>
4	-- Description: <Description,To fetch SalesOrderDetail>
5	-- ==
6	CREATE PROCEDURE [dbo].[usp_Get_SalesOrderDetail]
7	AS
8	BEGIN
9	-- SET NOCOUNT ON added to prevent extra result sets ...
10	-- interfering with SELECT statements.
11	SET NOCOUNT ON;
12	
13	-- Select statements for procedure here
14	SELECT top(100) SalesOrderID,SalesOrderDetailID,Carri...
15	END

Теперь, если вам нужно выбрать все записи из таблицы при выборе таблицы и нажать CTRL + 5 (вы можете выбрать любую клавишу). Вы можете сделать ярлык следующим образом.



Теперь идите и выберите имя таблицы из окна запроса и нажмите CTRL + 4 (выбранный нами ключ), это даст вам результат.

Прочитайте Клавиши быстрого доступа Microsoft SQL Server Management Studio онлайн: https://riptutorial.com/ru/sql-server/topic/7749/клавиши-быстрого-доступа-microsoft-sql-server-management-studio

глава 50: КЛАСТЕРИРОВАННАЯ КОЛОНКА

Examples

Таблица с индексом CLUSTERED COLUMNSTORE

Если вы хотите, чтобы таблица была организована в формате хранилища столбцов вместо хранилища строк, добавьте INDEX cci CLUSTERED COLUMNSTORE в определение таблицы:

```
DROP TABLE IF EXISTS Product
GO
CREATE TABLE Product (
    ProductID int,
    Name nvarchar(50) NOT NULL,
    Color nvarchar(15),
    Size nvarchar(5) NULL,
    Price money NOT NULL,
    Quantity int,
    INDEX cci CLUSTERED COLUMNSTORE
)
```

Таблицы COLUMSTORE лучше для таблиц, где вы ожидаете полного сканирования и отчетов, в то время как таблицы хранения строк лучше для таблиц, где вы будете читать или обновлять меньшие наборы строк.

Добавление кластерного индекса столбцов в существующую таблицу

CREATE CLUSTERED COLUMNSTORE INDEX позволяет организовать таблицу в формате столбцов:

```
DROP TABLE IF EXISTS Product
GO
CREATE TABLE Product (
    Name nvarchar(50) NOT NULL,
    Color nvarchar(15),
    Size nvarchar(5) NULL,
    Price money NOT NULL,
    Quantity int
)
GO
CREATE CLUSTERED COLUMNSTORE INDEX cci ON Product
```

Перестроить индекс CLUSTERED COLUMNSTORE

Индекс кластерного хранилища столбцов можно перестроить, если у вас много удаленных строк:

```
ALTER INDEX cci ON Products
REBUILD PARTITION = ALL
```

Восстановление CLUSTERED COLUMNSTORE «перезагрузит» данные из текущей таблицы в новую и снова применит сжатие, удалит удаленные строки и т. Д.

Вы можете перестроить один или несколько разделов.

Прочитайте КЛАСТЕРИРОВАННАЯ КОЛОНКА онлайн: https://riptutorial.com/ru/sql-server/topic/5774/кластерированная-колонка

глава 51: Ключевое слово Drop

Вступление

Ключевое слово Drop может использоваться с различными объектами SQL, в этом разделе приводятся быстрые примеры различного использования с объектами базы данных.

замечания

Ссылки на MSDN.

- ТАБЛИЦА DROP (Transact-SQL)
- ПРОЦЕДУРА ДРОЖ (Transact-SQL)
- DROP DATABASE (Transact-SQL)

Examples

Падение столов

Команда **DROP TABLE** удаляет определения таблиц и все данные, индексы, триггеры, ограничения и соответствующие разрешения.

Перед тем как отбросить таблицу, вы должны проверить, есть ли какой-либо объект (представления, хранимые процедуры и другие таблицы), которые ссылаются на таблицу.

Вы не можете отбросить таблицу, на которую ссылается другая таблица, с помощью FOREIGN KEY. Вы должны сначала сбросить ссылку FOREIGN KEY.

Вы можете отбросить таблицу, на которую ссылается представление или хранимая процедура, но после удаления таблицы вид или хранимая процедура больше не могут использоваться.

Синтаксис

```
DROP TABLE [ IF EXISTS ] [ database_name . [ schema_name ] . | schema_name . ]
table_name [ ,...n ] [ ; ]
```

- `IF EXISTS` - отбрасывать таблицу только в том случае, если существует
- `database_name` - указать имя базы данных, в которой содержится таблица
- `schema_name` - `schema_name` имя схемы, в которой находится таблица
- `table_name` - указать имя таблицы, подлежащей удалению.

Примеры

Удалите таблицу с именем **TABLE_1** из текущей базы данных и стандартной схемой dbo

```
DROP TABLE Table_1;
```

Удалите таблицу с **TABLE_1** из базы данных **HR** и схемы по умолчанию dbo

```
DROP TABLE HR.Table_1;
```

Удалите таблицу с **TABLE_1** из базы данных **HR** и **внешней** схемы

```
DROP TABLE HR.external.TABLE_1;
```

Падение баз данных

Команда **DROP DATABASE** удаляет каталог базы данных независимо от ее состояния (офлайн, только для чтения, подозреваемого и т. Д.) Из текущего экземпляра SQL Server.

База данных не может быть удалена, если с ней связаны снимки базы данных, так как сначала снижаются снимки базы данных.

Падение базы данных удаляет все файлы физического диска (если он не отключен), используемый базой данных, если вы не используете хранимую процедуру sp_detach_db.

Снимок моментального снимка базы данных удаляет моментальный снимок из экземпляра SQL Server и удаляет физические файлы, также используемые им.

Отбрасываемая база данных может быть восстановлена только путем восстановления резервной копии (а не из моментального снимка базы данных).

Синтаксис

```
DROP DATABASE [ IF EXISTS ] { database_name | database_snapshot_name } [ ,...n ] [;]
```

- `IF EXISTS` - отбрасывать таблицу только в том случае, если существует
- `database_name` - указывает имя базы данных, которое нужно удалить.
- `database_snapshot_name` - указывает снимок базы данных для удаления
-

Примеры

Удалите одну базу данных;

```
DROP DATABASE Database1;
```

Удаление нескольких баз данных

```
DROP DATABASE Database1, Database2;
```

Удаление моментального снимка

```
DROP DATABASE Database1_snapshot17;
```

Удаление, если база данных существует

```
DROP DATABASE IF EXISTS Database1;
```

Падение временных таблиц

В SQL-сервере у нас есть 2 типа временных таблиц:

1. `##GlobalTempTable` - это тип временной таблицы, которая просматривается между всеми сеансами пользователя.
2. `#LocalTempTable` temp tab - это тип временной таблицы, которая существует только в текущей области (только в реальном процессе - вы можете получить идентификатор вашего текущего процесса с помощью `SELECT @@SPID`)

Процесс разворачивания временных таблиц такой же, как и для обычной таблицы:

```
DROP TABLE [ database_name . [ schema_name ] . | schema_name . ] table_name
```

ПЕРЕД SQL Server 2016:

```
IF(OBJECT_ID('tempdb..#TempTable') is not null)
    DROP TABLE #TempTable;
```

SQL Server 2016:

```
DROP TABLE IF EXISTS #TempTable
```

Прочитайте Ключевое слово Drop онлайн: https://riptutorial.com/ru/sql-server/topic/9532/ключевое-слово-drop

глава 52: курсоры

Синтаксис

- DECLARE cursor_name CURSOR [LOCAL | ГЛОБАЛЬНЫЙ]
  - [FORWARD_ONLY | SCROLL]
    [STATIC | KEYSET | ДИНАМИКА | ПЕРЕМОТКА ВПЕРЕД]
    [READ_ONLY | SCROLL_LOCKS | ОПТИМИЗАЦИЯ]
    [TYPE_WARNING]
  - ДЛЯ select_statement
  - [FOR UPDATE [OF column_name [, ... n]]]

замечания

Обычно вы хотели бы избежать использования курсоров, поскольку они могут отрицательно влиять на производительность. Однако в некоторых особых случаях вам может потребоваться выполнить запись записи по записи и выполнить некоторые действия.

Examples

Основной прямой только курсор

Обычно вы хотели бы избежать использования курсоров, поскольку они могут отрицательно влиять на производительность. Однако в некоторых особых случаях вам может потребоваться выполнить запись записи по записи и выполнить некоторые действия.

```
DECLARE @orderId AS INT

-- here we are creating our cursor, as a local cursor and only allowing
-- forward operations
DECLARE rowCursor CURSOR LOCAL FAST_FORWARD FOR
    -- this is the query that we want to loop through record by record
    SELECT [OrderId]
    FROM [dbo].[Orders]

-- first we need to open the cursor
OPEN rowCursor

-- now we will initialize the cursor by pulling the first row of data, in this example the
[OrderId] column,
-- and storing the value into a variable called @orderId
FETCH NEXT FROM rowCursor INTO @orderId

-- start our loop and keep going until we have no more records to loop through
WHILE @@FETCH_STATUS = 0
```

```
BEGIN

    PRINT @orderId

    -- this is important, as it tells SQL Server to get the next record and store the
[OrderId] column value into the @orderId variable
    FETCH NEXT FROM rowCursor INTO @orderId

END

-- this will release any memory used by the cursor
CLOSE rowCursor
DEALLOCATE rowCursor
```

Синтаксис рудиментарного курсора

Простой синтаксис курсора, работающий на нескольких примерах тестовых строк:

```
/* Prepare test data */
DECLARE @test_table TABLE
(
    Id INT,
    Val VARCHAR(100)
);
INSERT INTO @test_table(Id, Val)
VALUES
    (1, 'Foo'),
    (2, 'Bar'),
    (3, 'Baz');
/* Test data prepared */

/* Iterator variable @myId, for example sake */
DECLARE @myId INT;

/* Cursor to iterate rows and assign values to variables */
DECLARE myCursor CURSOR FOR
    SELECT Id
    FROM @test_table;

/* Start iterating rows */
OPEN myCursor;
FETCH NEXT FROM myCursor INTO @myId;

/* @@FETCH_STATUS global variable will be 1 / true until there are no more rows to fetch */
WHILE @@FETCH_STATUS = 0
BEGIN

    /* Write operations to perform in a loop here. Simple SELECT used for example */
    SELECT Id, Val
    FROM @test_table
    WHERE Id = @myId;

    /* Set variable(s) to the next value returned from iterator; this is needed otherwise the
cursor will loop infinitely. */
    FETCH NEXT FROM myCursor INTO @myId;
END
/* After all is done, clean up */
CLOSE myCursor;
DEALLOCATE myCursor;
```

Результаты SSMS. Обратите внимание, что это все отдельные запросы, они никоим образом не унифицированы. Обратите внимание, как механизм запросов обрабатывает каждую итерацию один за другим, а не как набор.

Я бы	Val
1	Foo
(Затронуты 1 ряд)	
Я бы	**Val**
2	Бар
(Затронуты 1 ряд)	
Я бы	**Val**
3	Baz
(Затронуты 1 ряд)	

Прочитайте курсоры онлайн: https://riptutorial.com/ru/sql-server/topic/870/курсоры

глава 53: Логические функции

Examples

ВЫБИРАТЬ

SQL Server 2012

Возвращает элемент по указанному индексу из списка значений. Если `index` превышает границы `values` , возвращается `NULL` .

Параметры:

1. `index` : integer, index to item в `values` . 1 на основе.
2. `values` : любой тип, список, разделенный запятыми

```
SELECT CHOOSE (1, 'apples', 'pears', 'oranges', 'bananas') AS chosen_result

chosen_result
-------------
apples
```

IIF

SQL Server 2012

Возвращает одно из двух значений, в зависимости от того, оценивает ли данное булевское выражение значение true или false.

Параметры:

1. `boolean_expression` оценивается с тем, чтобы вернуть значение
2. `true_value` возвращается, если `boolean_expression` true
3. `false_value` возвращается, если `boolean_expression` значение false

```
SELECT IIF (42 > 23, 'I knew that!', 'That is not true.') AS iif_result

iif_result
------------
I knew that!
```

SQL Server 2012

`IIF` может быть заменен оператором `CASE` . В приведенном выше примере я напишу как

```
SELECT CASE WHEN 42 > 23 THEN 'I knew that!' ELSE 'That is not true.' END AS iif_result
```

```
iif_result
------------
I knew that!
```

Прочитайте Логические функции онлайн: https://riptutorial.com/ru/sql-server/topic/10647/логические-функции

глава 54: Магазин запросов

Examples

Включить хранилище запросов в базе данных

В базе данных можно включить хранилище запросов, используя следующую команду:

```
ALTER DATABASE tpch SET QUERY_STORE = ON
```

SQL Server / Azure SQL Database будет собирать информацию о выполненных запросах и предоставлять информацию в представлениях sys.query_store:

- sys.query_store_query
- sys.query_store_query_text
- sys.query_store_plan
- sys.query_store_runtime_stats
- sys.query_store_runtime_stats_interval
- sys.database_query_store_options
- sys.query_context_settings

Получить статистику выполнения для SQL-запросов / планов

Следующий запрос будет возвращать информацию о qeries, их планах и средней статистике относительно их продолжительности, времени процессора, физического и логического чтения io.

```
SELECT Txt.query_text_id, Txt.query_sql_text, Pl.plan_id,
        avg_duration, avg_cpu_time,
        avg_physical_io_reads, avg_logical_io_reads
FROM sys.query_store_plan AS Pl
JOIN sys.query_store_query AS Qry
    ON Pl.query_id = Qry.query_id
JOIN sys.query_store_query_text AS Txt
    ON Qry.query_text_id = Txt.query_text_id
JOIN sys.query_store_runtime_stats Stats
    ON Pl.plan_id = Stats.plan_id
```

Удаление данных из хранилища запросов

Если вы хотите удалить какой-либо запрос или план запроса из хранилища запросов, вы можете использовать следующие команды:

```
EXEC sp_query_store_remove_query 4;
EXEC sp_query_store_remove_plan 3;
```

Параметры для этих хранимых процедур - это идентификатор запроса / плана, полученный из системных представлений.

Вы также можете просто удалить статистику выполнения для конкретного плана, не удаляя план из магазина:

```
EXEC sp_query_store_reset_exec_stats 3;
```

Параметр, указанный для этого идентификатора плана процедуры.

Форсирование плана запроса

Оптимизатор SQL Query выберет возможный план baes, который он может найти для некоторого запроса. Если вы можете найти какой-то план, который оптимально подходит для какого-либо запроса, вы можете заставить QO всегда использовать этот план, используя следующую хранимую процедуру:

```
EXEC sp_query_store_unforce_plan @query_id, @plan_id
```

С этого момента QO всегда будет использовать план, предоставленный для запроса.

Если вы хотите удалить эту привязку, вы можете использовать следующую хранимую процедуру:

```
EXEC sp_query_store_force_plan @query_id, @plan_id
```

С этого момента QO снова попытается найти лучший план.

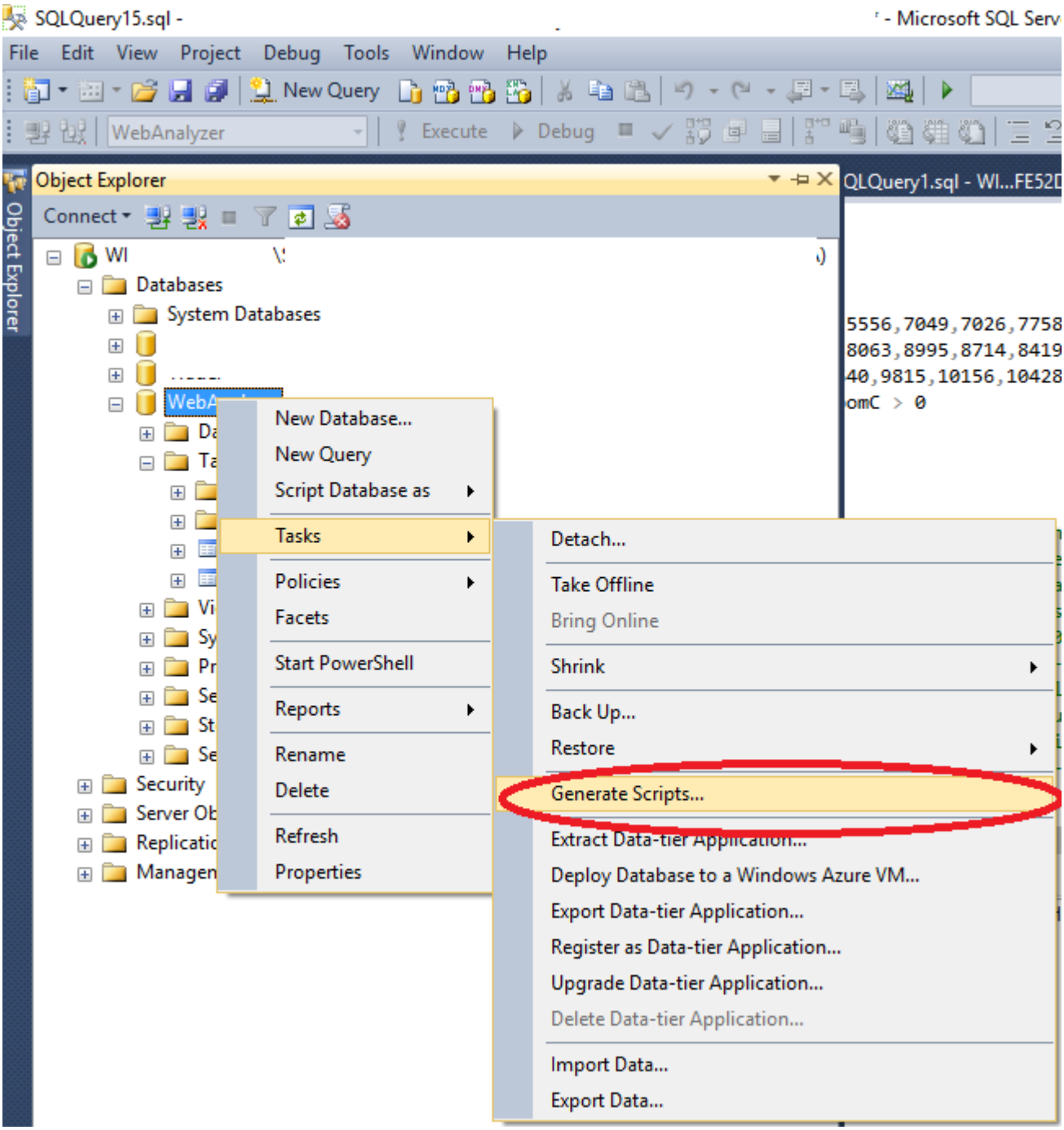
Прочитайте Магазин запросов онлайн: https://riptutorial.com/ru/sql-server/topic/7349/магазин-запросов

глава 55: миграция

Examples

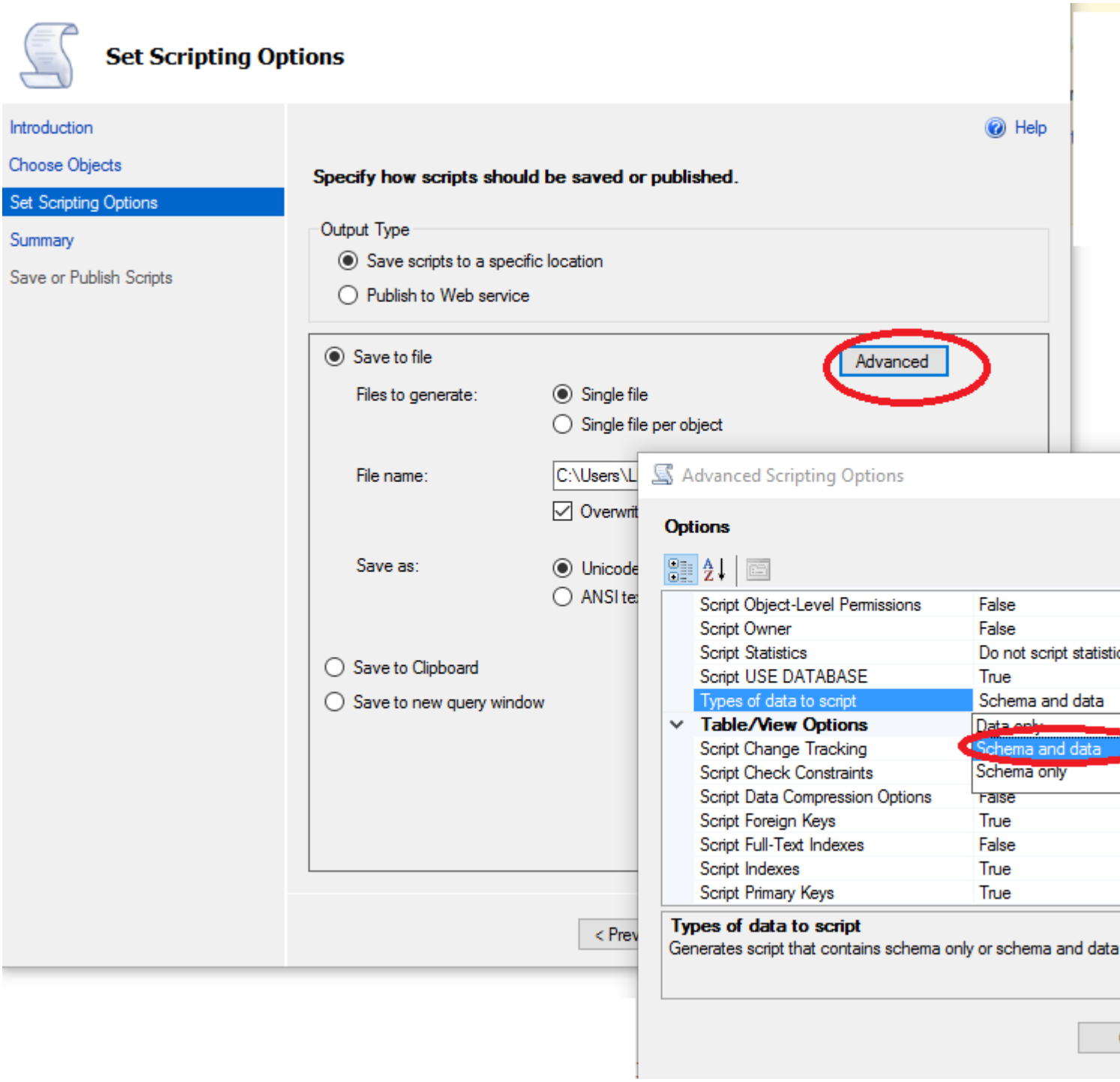
Как создать сценарии миграции

1. Щелкните `правой` кнопкой `мыши` на базе данных, которую вы хотите перенести затем -> `Tasks` -> `Generate Scripts...`



2. Мастер откроется, нажмите « `Next` затем выберите объекты, которые вы хотите перенести, и нажмите « `Next` раз, затем нажмите « `Advanced` прокрутите бит вниз и в «

`Types of data to script` выберите « `Schema and data` (если вы не хотите только структуры)



3. Нажмите пару еще раз « `Next` и « `Finish` и вы должны `.sql` свою базу данных в файле `.sql` .

4. запустите файл `.sql` на новом сервере, и вам нужно сделать это.

Прочитайте миграция онлайн: https://riptutorial.com/ru/sql-server/topic/4451/миграция

глава 56: Обозначение специальных символов и зарезервированных слов

замечания

Вообще говоря, лучше не использовать зарезервированные слова T-SQL как имена таблиц, имена столбцов, имена объектов программирования, псевдонимы и т. Д. Таким образом, способ избежать этих ключевых слов должен применяться только в том случае, если вы наследуете дизайн базы данных, который нельзя изменить ,

Для зарезервированных слов использование квадратных скобок не является обязательным. При использовании инструмента, такого как SQL Server Management Studio, зарезервированные слова будут выделены, чтобы привлечь внимание к тому, что они зарезервированы.

Examples

Основной метод

Основным методом выхода из зарезервированных слов для SQL Server является использование квадратных скобок (`[` и `]`). Например, « *Описание* и *имя»* являются зарезервированными словами; однако, если есть объект, использующий оба имени, используется синтаксис:

```
SELECT [Description]
FROM   dbo.TableName
WHERE  [Name] = 'foo'
```

Единственным специальным символом для SQL Server является одиночная кавычка `'` и он ускользает, удваивая его использование. Например, чтобы найти имя *O'Shea* в той же таблице, будет использоваться следующий синтаксис:

```
SELECT [Description]
FROM   dbo.TableName
WHERE  [Name] = 'O''Shea'
```

Прочитайте Обозначение специальных символов и зарезервированных слов онлайн: https://riptutorial.com/ru/sql-server/topic/7156/обозначение-специальных-символов-и-зарезервированных-слов

глава 57: Обработка транзакций

параметры

параметр	подробности
transaction_name	для наименования вашей транзакции - полезно с параметром [*с отметкой*], который позволит иметь смысл вносить в журнал - чувствительный к регистру (!)
с меткой [' description']	может быть добавлено к [*имя_контакта*] и будет хранить отметку в журнале

Examples

базовый скелет транзакций с обработкой ошибок

```
BEGIN TRY -- start error handling
    BEGIN TRANSACTION; -- from here on transactions (modifictions) are not final
        -- start your statement(s)
        select 42/0 as ANSWER  -- simple SQL Query with an error
        -- end your statement(s)
    COMMIT TRANSACTION; -- finalize all transactions (modifications)
END TRY   -- end error handling -- jump to end
BEGIN CATCH -- execute this IF an error occured
        ROLLBACK TRANSACTION; -- undo any transactions (modifications)
-- put together some information as a query
    SELECT
        ERROR_NUMBER() AS ErrorNumber
        ,ERROR_SEVERITY() AS ErrorSeverity
        ,ERROR_STATE() AS ErrorState
        ,ERROR_PROCEDURE() AS ErrorProcedure
        ,ERROR_LINE() AS ErrorLine
        ,ERROR_MESSAGE() AS ErrorMessage;

END CATCH;  -- final line of error handling
GO -- execute previous code
```

Прочитайте Обработка транзакций онлайн: https://riptutorial.com/ru/sql-server/topic/5859/обработка-транзакций

глава 58: Общие выражения таблицы

Синтаксис

- WITH *cte_name* [(*column_name_1* , *column_name_2* , ...)] AS (*cte_expression*)

замечания

Необходимо отделить CTE от предыдущего оператора с символом с запятой (;).

т.е. с помощью `;WITH CommonTableName (...) SELECT ... FROM CommonTableName ...`

Область CTE представляет собой единую партию и только ниже ее определения. Пакет может содержать несколько CTE, а CTE может ссылаться на другой CTE, определенный ранее в пакете, но CTE не может ссылаться на другой CTE, который определен позже в партии.

Examples

Иерархия сотрудников

Настройка таблицы

```
CREATE TABLE dbo.Employees
(
    EmployeeID INT NOT NULL PRIMARY KEY,
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    ManagerID INT NULL
)

GO

INSERT INTO Employees VALUES (101, 'Ken', 'Sánchez', NULL)
INSERT INTO Employees VALUES (102, 'Keith', 'Hall', 101)
INSERT INTO Employees VALUES (103, 'Fred', 'Bloggs', 101)
INSERT INTO Employees VALUES (104, 'Joseph', 'Walker', 102)
INSERT INTO Employees VALUES (105, 'Žydrė', 'Klybė', 101)
INSERT INTO Employees VALUES (106, 'Sam', 'Jackson', 105)
INSERT INTO Employees VALUES (107, 'Peter', 'Miller', 103)
INSERT INTO Employees VALUES (108, 'Chloe', 'Samuels', 105)
INSERT INTO Employees VALUES (109, 'George', 'Weasley', 105)
INSERT INTO Employees VALUES (110, 'Michael', 'Kensington', 106)
```

Выражение общей таблицы

```
;WITH cteReports (EmpID, FirstName, LastName, SupervisorID, EmpLevel) AS
(
    SELECT EmployeeID, FirstName, LastName, ManagerID, 1
    FROM Employees
    WHERE ManagerID IS NULL

    UNION ALL

    SELECT e.EmployeeID, e.FirstName, e.LastName, e.ManagerID, r.EmpLevel + 1
    FROM Employees        AS e
    INNER JOIN cteReports AS r ON e.ManagerID = r.EmpID
)

SELECT
    FirstName + ' ' + LastName AS FullName,
    EmpLevel,
    (SELECT FirstName + ' ' + LastName FROM Employees WHERE EmployeeID =
cteReports.SupervisorID) AS ManagerName
FROM cteReports
ORDER BY EmpLevel, SupervisorID
```

Выход:

ФИО	EmpLevel	ManagerName
Кен Санчес	1	*ноль*
Кит Холл	2	Кен Санчес
Fred Bloggs	2	Кен Санчес
Жидре Клыбе	2	Кен Санчес
Джозеф Уокер	3	Кит Холл
Питер Миллер	3	Fred Bloggs
Сэм Джексон	3	Жидре Клыбе
Хлоя Самуэльс	3	Жидре Клыбе
Джордж Уизли	3	Жидре Клыбе
Майкл Кенсингтон	4	Сэм Джексон

Найти nth самую высокую зарплату с помощью CTE

Таблица сотрудников:

```
|  ID  | FirstName | LastName | Gender | Salary |
```

```
+------+-----------+----------+--------+--------+
| 1    | Jahangir  | Alam     | Male   | 70000  |
| 2    | Arifur    | Rahman   | Male   | 60000  |
| 3    | Oli       | Ahammed  | Male   | 45000  |
| 4    | Sima      | Sultana  | Female | 70000  |
| 5    | Sudeepta  | Roy      | Male   | 80000  |
+------+-----------+----------+--------+--------+
```

CTE (общее выражение таблицы):

```
 WITH RESULT AS
(
    SELECT SALARY,
           DENSE_RANK() OVER (ORDER BY SALARY DESC) AS DENSERANK
    FROM EMPLOYEES
)
SELECT TOP 1 SALARY
FROM RESULT
WHERE DENSERANK = 1
```

Чтобы найти вторую самую высокую зарплату, просто замените N на 2. Аналогично, чтобы найти 3-ей самую высокую зарплату, просто замените N на 3.

Удаление повторяющихся строк с помощью CTE

Таблица сотрудников:

```
|  ID  | FirstName | LastName | Gender | Salary |
+------+-----------+----------+--------+--------+
|  1   | Mark      | Hastings | Male   | 60000  |
|  1   | Mark      | Hastings | Male   | 60000  |
|  2   | Mary      | Lambeth  | Female | 30000  |
|  2   | Mary      | Lambeth  | Female | 30000  |
|  3   | Ben       | Hoskins  | Male   | 70000  |
|  3   | Ben       | Hoskins  | Male   | 70000  |
|  3   | Ben       | Hoskins  | Male   | 70000  |
+------+-----------+----------+--------+--------+
```

CTE (общее выражение таблицы):

```
WITH EmployeesCTE AS
(
   SELECT *, ROW_NUMBER()OVER(PARTITION BY ID ORDER BY ID) AS RowNumber
   FROM Employees
)
DELETE FROM EmployeesCTE WHERE RowNumber > 1
```

Результат выполнения:

```
|  ID  | FirstName | LastName | Gender | Salary |
+------+-----------+----------+--------+--------+
|  1   | Mark      | Hastings | Male   | 60000  |
|  2   | Mary      | Lambeth  | Female | 30000  |
|  3   | Ben       | Hoskins  | Male   | 70000  |
```

```
+------+-----------+----------+--------+--------+
```

Создайте таблицу дат, используя CTE

```
DECLARE @startdate CHAR(8), @numberDays TINYINT

SET @startdate = '20160101'
SET @numberDays = 10;

WITH CTE_DatesTable
AS
(
  SELECT CAST(@startdate as date) AS [date]
  UNION ALL
  SELECT DATEADD(dd, 1, [date])
  FROM CTE_DatesTable
  WHERE DATEADD(dd, 1, [date]) <= DateAdd(DAY, @numberDays-1, @startdate)
)

SELECT [date] FROM CTE_DatesTable

OPTION (MAXRECURSION 0)
```

Этот пример возвращает таблицу столбцов с одной колонкой, начиная с даты, указанной в переменной @startdate, и возвращает следующую дату @numberDays.

Рекурсивный CTE

Этот пример показывает, как получить каждый год с этого года до 2011 года (2012 - 1).

```
WITH yearsAgo
(
    myYear
)
AS
(
     -- Base Case: This is where the recursion starts
     SELECT DATEPART(year, GETDATE()) AS myYear

     UNION ALL  -- This MUST be UNION ALL (cannot be UNION)

     -- Recursive Section: This is what we're doing with the recursive call
     SELECT yearsAgo.myYear - 1
     FROM yearsAgo
     WHERE yearsAgo.myYear >= 2012
)
     SELECT myYear FROM yearsAgo;  -- A single SELECT, INSERT, UPDATE, or DELETE
```

myYear
2016
2015

myYear
2014
2013
2012
2011

Вы можете управлять рекурсией (переполнение стека кода в коде) с помощью MAXRECURSION в качестве опции запроса, которая ограничивает количество рекурсивных вызовов.

```
WITH yearsAgo
(
    myYear
)
AS
(
     -- Base Case
     SELECT DATEPART(year , GETDATE()) AS myYear
     UNION ALL
     -- Recursive Section
     SELECT yearsAgo.myYear - 1
     FROM yearsAgo
     WHERE yearsAgo.myYear >= 2002
)
     SELECT * FROM yearsAgo
     OPTION (MAXRECURSION 10);
```

> Msg 530, уровень 16, состояние 1, строка 2Приложение завершено.
> Максимальная рекурсия 10 была исчерпана до завершения заявки.

CTE с несколькими операторами AS

```
;WITH cte_query_1
AS
(
    SELECT *
    FROM database.table1
),
cte_query_2
AS
(
    SELECT *
    FROM database.table2
)
SELECT *
FROM cte_query_1
WHERE cte_query_one.fk IN
(
    SELECT PK
    FROM cte_query_2
)
```

С помощью обычных табличных выражений можно создавать несколько запросов с использованием операторов AS, разделенных запятыми. Затем запрос может ссылаться на любой или все эти запросы разными способами, даже если они присоединяются к ним.

Прочитайте Общие выражения таблицы онлайн: https://riptutorial.com/ru/sql-server/topic/1343/общие-выражения-таблицы

глава 59: Ограничить набор результатов

Вступление

По мере роста таблиц базы данных часто бывает полезно ограничить результаты запросов фиксированным числом или процентом. Этого можно достичь с помощью ключевого слова `TOP` SQL или предложения `OFFSET FETCH` .

параметры

параметр	подробности
`TOP`	Ограничить ключевое слово. Используйте с числом.
`PERCENT`	Процентное ключевое слово. Поставляется после `TOP` и предельного числа.

замечания

Если используется предложение `ORDER BY` , ограничение применяется к упорядоченному набору результатов.

Examples

Ограничение с помощью TOP

Этот пример ограничивает результат `SELECT` до 100 строк.

```
SELECT TOP 100 *
FROM table_name;
```

Также можно использовать переменную, чтобы указать количество строк:

```
DECLARE @CountDesiredRows int = 100;
SELECT TOP (@CountDesiredRows) *
FROM table_name;
```

Ограничение с помощью PERCENT

Этот пример ограничивает результат `SELECT` до 15 процентов от общего количества строк.

```
SELECT TOP 15 PERCENT *
FROM table_name
```

Ограничение с помощью FETCH

SQL Server 2012

`FETCH` обычно более полезен для разбивки на страницы, но может использоваться как альтернатива `TOP` :

```
SELECT *
FROM table_name
ORDER BY 1
OFFSET 0 ROWS
FETCH NEXT 50 ROWS ONLY
```

Прочитайте Ограничить набор результатов онлайн: https://riptutorial.com/ru/sql-server/topic/1555/ограничить-набор-результатов

глава 60: Оператор SELECT

Вступление

В SQL `SELECT` возвращают наборы результатов из коллекций данных, таких как таблицы или представления. Операторы `SELECT` могут использоваться с различными другими предложениями, такими как `WHERE` , `GROUP BY` или `ORDER BY` для дальнейшего уточнения желаемых результатов.

Examples

Basic SELECT из таблицы

Выберите все столбцы из какой-либо таблицы (в этом случае системная таблица):

```
SELECT *
FROM sys.objects
```

Или выберите только некоторые конкретные столбцы:

```
SELECT object_id, name, type, create_date
FROM sys.objects
```

Фильтровать строки, используя предложение WHERE

Предложение WHERE фильтрует только те строки, которые удовлетворяют некоторому условию:

```
SELECT *
FROM sys.objects
WHERE type = 'IT'
```

Сортировка результатов с помощью ORDER BY

Предложение ORDER BY сортирует строки в возвращаемом результирующем наборе с помощью некоторого столбца или выражения:

```
SELECT *
FROM sys.objects
ORDER BY create_date
```

Результат группы с использованием GROUP BY

Предложение GROUP BY группирует строки по некоторому значению:

```
SELECT type, count(*) as c
FROM sys.objects
GROUP BY type
```

Вы можете применить некоторую функцию для каждой группы (агрегированную функцию) для вычисления суммы или количества записей в группе.

тип	с
SQ	3
S	72
ЭТО	16
PK	1
U	5

Группы фильтров с использованием предложения HAVING

Предложение HAVING удаляет группы, которые не удовлетворяют условию:

```
SELECT type, count(*) as c
FROM sys.objects
GROUP BY type
HAVING count(*) < 10
```

тип	с
SQ	3
PK	1
U	5

Возвращение только первых N строк

Предложение TOP возвращает только первые N строк в результате:

```
SELECT TOP 10 *
FROM sys.objects
```

Разбиение страницы с помощью OFFSET FETCH

Предложение OFFSET FETCH - это более продвинутая версия TOP. Он позволяет пропустить строки N1 и взять следующие строки N2:

```
SELECT *
FROM sys.objects
ORDER BY object_id
OFFSET 50 ROWS FETCH NEXT 10 ROWS ONLY
```

Вы можете использовать OFFSET без выборки, чтобы просто пропустить первые 50 строк:

```
SELECT *
FROM sys.objects
ORDER BY object_id
OFFSET 50 ROWS
```

SELECT без FROM (нет данных)

Оператор SELECT может быть выполнен без предложения FROM:

```
declare @var int = 17;

SELECT @var as c1, @var + 2 as c2, 'third' as c3
```

В этом случае возвращается одна строка со значениями / результатами выражений.

Прочитайте Оператор SELECT онлайн: https://riptutorial.com/ru/sql-server/topic/4662/оператор-select

глава 61: Опция Ties

Examples

Данные испытаний

```
CREATE TABLE #TEST
(
Id INT,
Name VARCHAR(10)
)

Insert Into #Test
select 1,'A'
Union All
Select 1,'B'
union all
Select 1,'C'
union all
Select 2,'D'
```

Ниже приведен результат таблицы выше. Как вы можете видеть, Id Column повторяется три раза.

```
Id   Name
1    A
1    B
1    C
2    D
```

Теперь давайте проверим выход, используя простой порядок.

```
Select Top (1)  Id,Name From
#test
Order By Id ;
```

Выходные данные: (Результат запроса выше не гарантируется одинаковым с каждым разом)

```
Id   Name
1    B
```

Позволяет запустить тот же запрос с опцией «Связи».

```
Select Top (1) With Ties Id,Name
 From
#test
Order By Id
```

Выход :

```
Id   Name
1    A
1    B
1    C
```

Как вы можете видеть, SQL Server выводит все строки **, привязанные к** Order by Column. Давайте посмотрим еще один пример, чтобы понять это лучше.

```
Select Top (1) With Ties Id,Name
 From
#test
Order By Id ,Name
```

Выход:

```
Id   Name
1    A
```

В резюме, когда мы используем параметр Ties Option, SQL Server выводит все строки Tied, независимо от того, какой лимит мы налагаем

Прочитайте Опция Ties онлайн: https://riptutorial.com/ru/sql-server/topic/2546/опция-ties

глава 62: Основные операции DDL в MS SQL Server

Examples

Начиная

В этом разделе описываются некоторые базовые команды **DDL** (= « **D** ata **D** efinition **L** anguage») для создания базы данных, таблицы в базе данных, представления и, наконец, хранимой процедуры.

Создать базу данных

Следующая команда SQL создает новую базу данных `Northwind` на текущем сервере, используя путь `C:\Program Files\Microsoft SQL Server\MSSQL11.INSTSQL2012\MSSQL\DATA\` :

```
USE [master]
GO

CREATE DATABASE [Northwind]
 CONTAINMENT = NONE
 ON  PRIMARY
 (
  NAME = N'Northwind',
  FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL11.INSTSQL2012\MSSQL\DATA\Northwind.mdf' , SIZE = 5120KB , MAXSIZE = UNLIMITED,
FILEGROWTH = 1024KB
 )
 LOG ON
 (
  NAME = N'Northwind_log',
  FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL11.INSTSQL2012\MSSQL\DATA\Northwind_log.ldf' , SIZE = 1536KB , MAXSIZE = 2048GB ,
FILEGROWTH = 10%
 )
GO

ALTER DATABASE [Northwind] SET COMPATIBILITY_LEVEL = 110
GO
```

Примечание. База данных T-SQL состоит из двух файлов: файла базы данных `*.mdf` и журнала транзакций `*.ldf` . Оба должны быть указаны при создании новой базы данных.

Создать таблицу

Следующая команда SQL создает новую таблицу `Categories` в текущей базе данных, используя схему `dbo` (вы можете переключить контекст базы данных с помощью `Use <DatabaseName>`):

```
CREATE TABLE dbo.Categories(
    CategoryID int IDENTITY NOT NULL,
    CategoryName nvarchar(15) NOT NULL,
    Description ntext NULL,
    Picture image NULL,
      CONSTRAINT PK_Categories PRIMARY KEY CLUSTERED
      (
         CategoryID ASC
      )
      WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
            ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON PRIMARY
) ON PRIMARY TEXTIMAGE_ON PRIMARY
```

Создать представление

Следующая команда SQL создает новое представление `Summary_of_Sales_by_Year` в текущей базе данных, используя схему `dbo` (вы можете переключить контекст базы данных с помощью `Use <DatabaseName>`):

```
CREATE VIEW dbo.Summary_of_Sales_by_Year AS
  SELECT ord.ShippedDate, ord.OrderID, ordSub.Subtotal
  FROM Orders ord
  INNER JOIN [Order Subtotals] ordSub ON ord.OrderID = ordSub.OrderID
```

Это приведет к объединению таблиц « `Orders` и `[Order Subtotals]` `ShippedDate` `[Order Subtotals]` для отображения столбцов `ShippedDate` , `OrderID` и `Subtotal` . Поскольку таблица `[Order Subtotals]` имеет пробел в своем имени в базе данных Northwind, ее необходимо заключить в квадратные скобки.

Создать процедуру

Следующая команда SQL создает новую хранимую процедуру `CustOrdersDetail` в текущей базе данных, используя схему `dbo` (вы можете переключить контекст базы данных с помощью `Use <DatabaseName>`):

```
CREATE PROCEDURE dbo.MyCustOrdersDetail @OrderID int, @MinQuantity int=0
AS BEGIN
  SELECT ProductName,
    UnitPrice=ROUND(Od.UnitPrice, 2),
    Quantity,
    Discount=CONVERT(int, Discount * 100),
    ExtendedPrice=ROUND(CONVERT(money, Quantity * (1 - Discount) * Od.UnitPrice), 2)
  FROM Products P, [Order Details] Od
  WHERE Od.ProductID = P.ProductID and Od.OrderID = @OrderID
```

```
  and Od.Quantity>=@MinQuantity
END
```

Эта хранимая процедура после ее создания может быть вызвана следующим образом:

```
exec dbo.MyCustOrdersDetail 10248
```

который будет возвращать все детали заказа с @ OrderId = 10248 (и количеством> = 0 по умолчанию). Или вы можете указать необязательный параметр

```
exec dbo.MyCustOrdersDetail 10248, 10
```

который будет возвращать только заказы с минимальным количеством 10 (или более).

Прочитайте Основные операции DDL в MS SQL Server онлайн: https://riptutorial.com/ru/sql-server/topic/5463/основные-операции-ddl-в-ms-sql-server

глава 63: пагинация

Вступление

Смещение строк и пейджинг в разных версиях SQL Server

Синтаксис

- SELECT * FROM TableName ORDER BY id OFFSET 10 ROWS FETCH NEXT 10 ROWS ONLY;

Examples

Разбиение на страницы с использованием ROW_NUMBER с общим выражением таблицы

SQL Server 2008

Функция `ROW_NUMBER` может назначать инкрементный номер каждой строке в результирующем наборе. В сочетании с общим выражением таблицы , использующим оператор `BETWEEN` , можно создавать «страницы» наборов результатов. Например: страница 1, содержащая результаты 1-10, вторая страница содержит результаты 11-20, стр. 3, содержащие результаты 21-30 и т. Д.

```
WITH data
AS
(
    SELECT ROW_NUMBER() OVER (ORDER BY name) AS row_id,
        object_id,
        name,
        type,
        create_date
    FROM sys.objects
)
SELECT *
FROM data
WHERE row_id BETWEEN 41 AND 50
```

Примечание. Невозможно использовать `ROW_NUMBER` в предложении `WHERE` , например:

```
SELECT object_id,
    name,
    type,
    create_date
FROM sys.objects
WHERE ROW_NUMBER() OVER (ORDER BY name) BETWEEN 41 AND 50
```

Хотя это было бы более удобно, SQL Server вернет следующую ошибку в этом случае:

> Msg 4108, уровень 15, состояние 1, строка 6
>
> Оконные функции могут отображаться только в предложениях SELECT или ORDER BY.

Разбиение страницы с помощью OFFSET FETCH

SQL Server 2012

Предложение `OFFSET FETCH` реализует разбиение на страницы более кратким образом. С его помощью можно пропустить строки N1 (указанные в `OFFSET`) и вернуть следующие строки N2 (указанные в `FETCH`):

```
SELECT *
FROM sys.objects
ORDER BY object_id
OFFSET 40 ROWS FETCH NEXT 10 ROWS ONLY
```

Предложение `ORDER BY` требуется для обеспечения детерминированных результатов.

Paginaton с внутренним запросом

В более ранних версиях SQL Server разработчикам приходилось использовать двойную сортировку в сочетании с ключевым словом `TOP` чтобы возвращать строки на странице:

```
 SELECT TOP 10 *
 FROM
 (
    SELECT
    TOP 50 object_id,
        name,
        type,
        create_date
    FROM sys.objects
    ORDER BY name ASC
) AS data
ORDER BY name DESC
```

Внутренний запрос вернет первые 50 строк, упорядоченных по `name` . Затем внешний запрос изменит порядок этих 50 строк и выберет первые 10 строк (это будут последние 10 строк в группе до разворота).

Пейджинг в различных версиях SQL Server

SQL Server 2012/2014

```
DECLARE @RowsPerPage INT = 10, @PageNumber INT = 4
SELECT OrderId, ProductId
FROM OrderDetail
ORDER BY OrderId
OFFSET (@PageNumber - 1) * @RowsPerPage ROWS
FETCH NEXT @RowsPerPage ROWS ONLY
```

SQL Server 2005/2008 / R2

```
DECLARE @RowsPerPage INT = 10, @PageNumber INT = 4
SELECT OrderId, ProductId
FROM (
    SELECT OrderId, ProductId, ROW_NUMBER() OVER (ORDER BY OrderId) AS RowNum
    FROM OrderDetail) AS OD
WHERE OD.RowNum BETWEEN ((@PageNumber - 1 ) * @RowsPerPage) + 1
AND @RowsPerPage * @PageNumber
```

SQL Server 2000

```
DECLARE @RowsPerPage INT = 10, @PageNumber INT = 4
SELECT OrderId, ProductId
FROM (SELECT TOP (@RowsPerPage) OrderId, ProductId
        FROM (SELECT TOP ((@PageNumber)*@RowsPerPage) OrderId, ProductId
                FROM OrderDetail
                ORDER BY OrderId) AS OD
    ORDER BY OrderId DESC) AS OD2
ORDER BY OrderId ASC
```

SQL Server 2012/2014 с использованием ORDER BY OFFSET и FETCH NEXT

Для получения следующих 10 строк просто запустите этот запрос:

```
SELECT * FROM TableName ORDER BY id OFFSET 10 ROWS FETCH NEXT 10 ROWS ONLY;
```

Ключевые моменты, которые следует учитывать при его использовании:

- `ORDER BY` является обязательным для использования предложений `OFFSET` и `FETCH` .
- Предложение `OFFSET` является обязательным для `FETCH` . Вы никогда не сможете использовать `ORDER BY ... FETCH` .
- `TOP` нельзя комбинировать с `OFFSET` и `FETCH` в одном выражении запроса.

Прочитайте пагинация онлайн: https://riptutorial.com/ru/sql-server/topic/6874/пагинация

глава 64: Первичные ключи

замечания

Первичные ключи используются для уникальной идентификации записи в таблице. В таблице может быть только один первичный ключ (хотя первичный ключ может состоять из нескольких столбцов), а для определенных типов репликации требуется первичный ключ.

Первичные ключи часто используются как (но не обязательно) кластеризованный индекс в таблице.

Examples

Создать стол w / идентификационный столбец в качестве первичного ключа

```
-- Identity primary key - unique arbitrary increment number
create table person (
id int identity(1,1) primary key not null,
firstName varchar(100) not null,
lastName varchar(100) not null,
dob DateTime not null,
ssn varchar(9) not null
)
```

Создать таблицу с первичным ключом GUID

```
-- GUID primary key - arbitrary unique value for table
create table person (
id uniqueIdentifier default (newId()) primary key,
firstName varchar(100) not null,
lastName varchar(100) not null,
dob DateTime not null,
ssn varchar(9) not null
)
```

Создать таблицу с естественным ключом

```
-- natural primary key - using an existing piece of data within the table that uniquely
identifies the record
create table person (
firstName varchar(100) not null,
lastName varchar(100) not null,
dob DateTime not null,
ssn varchar(9) primary key not null
)
```

Создать таблицу с составным ключом

```
-- composite key - using two or more existing columns within a table to create a primary key
create table person (
firstName varchar(100) not null,
lastName varchar(100) not null,
dob DateTime not null,
ssn varchar(9) not null,
primary key (firstName, lastName, dob)
)
```

Добавить первичный ключ в существующую таблицу

```
ALTER TABLE person
 ADD CONSTRAINT pk_PersonSSN PRIMARY KEY (ssn)
```

Обратите внимание: если столбец первичного ключа (в данном случае `ssn`) имеет более одной строки с одним и тем же ключом-кандидатом, вышеуказанный оператор будет терпеть неудачу, поскольку значения первичного ключа должны быть уникальными.

Удалить первичный ключ

```
ALTER TABLE Person
 DROP CONSTRAINT pk_PersonSSN
```

Прочитайте Первичные ключи онлайн: https://riptutorial.com/ru/sql-server/topic/4543/первичные-ключи

глава 65: переменные

Синтаксис

- DECLARE @VariableName DataType [= Value];
- SET @VariableName = Value;

Examples

Объявление переменной таблицы

```
DECLARE @Employees TABLE
(
    EmployeeID INT NOT NULL PRIMARY KEY,
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    ManagerID INT NULL
)
```

Когда вы создаете обычную таблицу, вы используете синтаксис `CREATE TABLE Name (Columns)` . При создании переменной таблицы используется `DECLARE @Name TABLE (Columns)` .

Чтобы ссылаться на переменную таблицы внутри `SELECT` , SQL Server требует, чтобы вы указали переменную таблицы как псевдоним, иначе вы получите сообщение об ошибке:

> Должен объявить скалярную переменную «@TableVariableName».

т.е.

```
DECLARE @Table1 TABLE (Example INT)
DECLARE @Table2 TABLE (Example INT)

/*
-- the following two commented out statements would generate an error:
SELECT *
FROM @Table1
INNER JOIN @Table2 ON @Table1.Example = @Table2.Example

SELECT *
FROM @Table1
WHERE @Table1.Example = 1
*/

-- but these work fine:
SELECT *
FROM @Table1 T1
INNER JOIN @Table2 T2 ON T1.Example = T2.Example

SELECT *
FROM @Table1 Table1
```

```
WHERE Table1.Example = 1
```

Обновление переменной с помощью SET

```
DECLARE @VariableName INT
SET @VariableName = 1
PRINT @VariableName
```

1

Используя `SET` , вы можете обновлять только одну переменную за раз.

Обновление переменных с помощью SELECT

Используя `SELECT` , вы можете обновлять сразу несколько переменных.

```
DECLARE @Variable1 INT, @Variable2 VARCHAR(10)
SELECT @Variable1 = 1, @Variable2 = 'Hello'
PRINT @Variable1
PRINT @Variable2
```

1

Привет

При использовании `SELECT` для обновления переменной из столбца таблицы, если имеется несколько значений, она будет использовать *последнее* значение. (Применяются правила нормального порядка - если сортировка не задана, заказ не гарантируется).

```
CREATE TABLE #Test (Example INT)
INSERT INTO #Test VALUES (1), (2)

DECLARE @Variable INT
SELECT @Variable = Example
FROM #Test
ORDER BY Example ASC

PRINT @Variable
```

2

```
SELECT TOP 1 @Variable = Example
FROM #Test
ORDER BY Example ASC

PRINT @Variable
```

1

Если в запросе нет строк, значение переменной не изменится:

```
SELECT TOP 0 @Variable = Example
FROM #Test
ORDER BY Example ASC

PRINT @Variable
```

1

Объявлять сразу несколько переменных с начальными значениями

```
DECLARE
  @Var1 INT = 5,
  @Var2 NVARCHAR(50) = N'Hello World',
  @Var3 DATETIME = GETDATE()
```

Операторы вспомогательного присваивания

SQL Server 2008 R2

Поддерживаемые составные операторы:

- `+=` Добавить и назначить
- `-=` Вычесть и назначить
- `*=` Умножить и назначить
- `/=` Разделить и назначить
- `%=` Modulo и назначить
- `&=` Побитовое И и назначить
- `^=` Побитовое XOR и назначить
- `|=` Побитовое ИЛИ и назначить

Пример использования:

```
DECLARE @test INT = 42;
SET @test += 1;
PRINT @test;    --43
SET @test -= 1;
PRINT @test;    --42
SET @test *= 2
PRINT @test;    --84
SET @test /= 2;
PRINT @test;    --42
```

Обновление переменных путем выбора из таблицы

В зависимости от структуры ваших данных вы можете создавать переменные, которые обновляются динамически.

```
DECLARE @CurrentID int = (SELECT TOP 1 ID FROM Table ORDER BY CreateDate desc)


DECLARE @Year int = 2014
DECLARE @CurrentID int = (SELECT ID FROM Table WHERE Year = @Year)
```

В большинстве случаев вы захотите убедиться, что ваш запрос возвращает только одно значение при использовании этого метода.

Прочитайте переменные онлайн: https://riptutorial.com/ru/sql-server/topic/2566/переменные

глава 66: Перемещение и копирование данных вокруг таблиц

Examples

Копирование данных из одной таблицы в другую

Этот код выбирает данные из таблицы и отображает их в инструменте запроса (обычно SSMS)

```
SELECT Column1, Column2, Column3 FROM MySourceTable;
```

Этот код вставляет эти данные в таблицу:

```
INSERT INTO MyTargetTable (Column1, Column2, Column3)
SELECT Column1, Column2, Column3 FROM MySourceTable;
```

Копирование данных в таблицу, создание этой таблицы на лету

Этот код выбирает данные из таблицы:

```
SELECT Column1, Column2, Column3 FROM MySourceTable;
```

Этот код создает новую таблицу `MyNewTable` и помещает в нее эти данные

```
SELECT Column1, Column2, Column3
INTO MyNewTable
FROM MySourceTable;
```

Перемещение данных в таблицу (при использовании метода уникальных ключей)

Чтобы *перенести* данные, вы сначала вставляете их в цель, а затем удаляете все, что вы вставили из исходной таблицы. Это не нормальная операция SQL, но она может быть просветляющей

Что вы вставляли? Обычно в базах данных необходимо иметь один или несколько столбцов, которые можно использовать для однозначной идентификации строк, поэтому мы будем предполагать это и использовать его.

Этот оператор выбирает некоторые строки

```
SELECT Key1, Key2, Column3, Column4 FROM MyTable;
```

Сначала мы вставляем их в нашу целевую таблицу:

```
INSERT INTO TargetTable (Key1, Key2, Column3, Column4)
SELECT Key1, Key2, Column3, Column4 FROM MyTable;
```

Теперь, *предполагая, что записи в обеих таблицах уникальны* в `Key1` , `Key2` , мы можем использовать это для поиска и удаления данных из исходной таблицы

```
DELETE MyTable
WHERE EXISTS (
    SELECT * FROM TargetTable
    WHERE TargetTable.Key1 = SourceTable.Key1
    AND TargetTable.Key2 = SourceTable.Key2
);
```

Это будет работать только в том `Key1` , если `Key1` , `Key2` уникальны в обеих таблицах

Наконец, мы не хотим, чтобы работа была выполнена наполовину. Если мы завершим это в транзакции, то либо все данные будут перемещены, либо ничего не произойдет. Это гарантирует, что мы не вставляем данные, а затем не можем удалить данные из источника.

```
BEGIN TRAN;

INSERT INTO TargetTable (Key1, Key2, Column3, Column4)
SELECT Key1, Key2, Column3, Column4 FROM MyTable;


DELETE MyTable
WHERE EXISTS (
    SELECT * FROM TargetTable
    WHERE TargetTable.Key1 = SourceTable.Key1
    AND TargetTable.Key2 = SourceTable.Key2
);

COMMIT TRAN;
```

Прочитайте Перемещение и копирование данных вокруг таблиц онлайн: https://riptutorial.com/ru/sql-server/topic/1467/перемещение-и-копирование-данных-вокруг-таблиц

глава 67: подзапросов

Examples

подзапросов

Подзапрос - это запрос в другом SQL-запросе. Подзапрос также называется внутренним запросом или внутренним выбором, а оператор, содержащий подзапрос, называется внешним запросом или внешним выбором.

Заметка

1. Подзапросы должны быть заключены в круглые скобки,
2. ORDER BY нельзя использовать в подзапросе.
3. Тип изображения, такой как BLOB, массив, текстовые типы данных, не разрешен в подзапросах.

Подзапросы могут использоваться с оператором select, insert, update и delete, в котором, from, select, а также IN, операторы сравнения и т. Д.

У нас есть таблица с именем ITCompanyInNepal, на которой мы будем выполнять запросы, чтобы показать примеры подзапросов:

ID	CompanyName	CompanyAddress	Headquarter	NumberOfEmployee
1	CompanyOne	Kathmandu	USA	350
2	CompanyTwo	Kathmandu	USA	310
3	CompanyThree	Kathmandu	Nepal	300
4	CompanyFour	Kathmandu	Nepal	180
5	CompanyFive	Birgunj	Denmark	150
6	CompanySix	Janakpur	USA	100
7	CompanySeven	Janakpur	Australia	100
8	CompanyEight	Birganj	Australia	150
9	CompanyNine	Biratnagar	Canada	200
10	CompanyTen	Pokhara	India	85

Примеры: **SubQueries с инструкцией Select**

с оператором **In** и **где** :

```
SELECT *
FROM ITCompanyInNepal
WHERE Headquarter IN (SELECT Headquarter
                      FROM ITCompanyInNepal
                      WHERE Headquarter = 'USA');
```

с **оператором сравнения** и **где** условие

```
SELECT *
FROM ITCompanyInNepal
WHERE NumberOfEmployee < (SELECT AVG(NumberOfEmployee)
                          FROM ITCompanyInNepal
                        )
```

с предложением **select**

```
SELECT   CompanyName,
         CompanyAddress,
         Headquarter,
         (Select SUM(NumberOfEmployee)
         FROM ITCompanyInNepal
         Where Headquarter = 'USA') AS TotalEmployeeHiredByUSAInKathmandu
FROM     ITCompanyInNepal
WHERE    CompanyAddress = 'Kathmandu' AND Headquarter = 'USA'
```

Подзапросы со вставкой

Нам нужно вставить данные из таблицы IndianCompany в ITCompanyInNepal. Таблица для IndianCompany приведена ниже:

ID	CompanyName	CompanyAddress	Headquarter	NumberOfEmployee
1	CompanyA	Banglore	USA	450
2	CompanyB	Banglore	USA	500
3	CompanyC	Hyderabad	Denmark	480
4	CompanyD	Hyderabad	Australia	780
5	CompanyE	Delhi	Canada	790

```
INSERT INTO ITCompanyInNepal
SELECT *
FROM IndianCompany
```

Подзапросы с оператором обновления

Предположим, что все компании, штаб-квартира которых находится в США, решили уволить 50 сотрудников из всех американских компаний в Непале из-за некоторых изменений в политике компаний США.

```
UPDATE ITCompanyInNepal
SET NumberOfEmployee = NumberOfEmployee - 50
WHERE Headquarter IN (SELECT Headquarter
                      FROM ITCompanyInNepal
                      WHERE Headquarter = 'USA')
```

Подзапросы с инструкцией Delete

Предположим, что все компании, штаб-квартира которых является Данией, решили закрыть свои компании из Непала.

```
DELETE FROM ITCompanyInNepal
```

```
WHERE Headquarter IN (SELECT Headquarter
                      FROM ITCompanyInNepal
                      WHERE Headquarter = 'Denmark')
```

Прочитайте подзапросов онлайн: https://riptutorial.com/ru/sql-server/topic/5629/подзапросов

глава 68: Подсказки по запросам

Examples

СОВЕТЫ СОВЕТОВ

Когда вы присоединяетесь к двум таблицам, оптимизатор запросов SQL Server (QO) может выбирать разные типы соединений, которые будут использоваться в запросе:

- Подключение HASH
- Соединение LOOP
- Присоединение MERGE

QO исследует планы и выбирает оптимального оператора для объединения таблиц. Однако, если вы уверены, что знаете, что будет оптимальным оператором объединения, вы можете указать, какой тип JOIN следует использовать. Внутреннее объединение LOOP заставит QO выбрать объединение вложенных циклов при соединении двух таблиц:

```
select top 100 *
from Sales.Orders o
    inner loop join Sales.OrderLines ol
    on o.OrderID = ol.OrderID
```

внутреннее объединение слияния приведет к соединению MERGE:

```
select top 100 *
from Sales.Orders o
    inner merge join Sales.OrderLines ol
    on o.OrderID = ol.OrderID
```

внутреннее хеш-соединение заставит оператор HASH-соединения:

```
select top 100 *
from Sales.Orders o
    inner hash join Sales.OrderLines ol
    on o.OrderID = ol.OrderID
```

ГРУППА ПО СОВЕТОМ

Когда вы используете предложение GROUP BY, оптимизатор запросов SQL Server (QO) может выбирать различные типы операторов группировки:

- HASH Aggregate, создающий хэш-карту для группировки записей
- Stream Aggregate, который хорошо работает с заранее заказанными входами

Вы можете явно потребовать, чтобы QO выбирает тот или иной агрегированный оператор,

если вы знаете, что было бы оптимальным. С помощью OPTION (ORDER GROUP) QO всегда будет выбирать агрегат Stream и добавить оператор Sort перед агрегатом Stream, если вход не отсортирован:

```
select OrderID, AVG(Quantity)
from Sales.OrderLines
group by OrderID
OPTION (ORDER GROUP)
```

С помощью OPTION (HASH GROUP) QO всегда будет выбирать агрегат Hash:

```
select OrderID, AVG(Quantity)
from Sales.OrderLines
group by OrderID
OPTION (HASH GROUP)
```

Подсказка FAST rows

Указывает, что запрос оптимизирован для быстрого извлечения первого number_rows. Это неотрицательное целое число. После того, как будут возвращены первые number_rows, запрос продолжит выполнение и даст полный набор результатов.

```
select OrderID, AVG(Quantity)
from Sales.OrderLines
group by OrderID
OPTION (FAST 20)
```

Советы UNION

Когда вы используете оператор UNION для двух результатов запроса, оптимизатор запросов (QO) может использовать следующие операторы для создания объединения двух наборов результатов:

- Объединение (Союз)
- Конкат (Союз)
- Hash Match (Союз)

Вы можете явно указать, какой оператор следует использовать с помощью подсказки OPTION ():

```
select OrderID, OrderDate, ExpectedDeliveryDate, Comments
from Sales.Orders
where OrderDate > DATEADD(day, -1, getdate())
UNION
select PurchaseOrderID as OrderID, OrderDate, ExpectedDeliveryDate, Comments
from Purchasing.PurchaseOrders
where OrderDate > DATEADD(day, -1, getdate())
OPTION(HASH UNION)
-- or OPTION(CONCAT UNION)
```

```
-- or OPTION(MERGE UNION)
```

Опция MAXDOP

Указывает максимальную степень параллелизма для запроса с указанием этой опции.

```
SELECT OrderID,
    AVG(Quantity)
FROM Sales.OrderLines
GROUP BY OrderID
OPTION (MAXDOP 2);
```

Этот параметр переопределяет параметр конфигурации MAXDOP для sp_configure и Resource Governor. Если MAXDOP установлен на ноль, сервер выбирает максимальную степень параллелизма.

Подсказки INDEX

Индексные подсказки используются, чтобы заставить запрос использовать определенный индекс, вместо того, чтобы позволить оптимизатору запросов SQL Server выбирать то, что он считает лучшим индексом. В некоторых случаях вы можете получить преимущества, указав индекс, который должен использовать запрос. Обычно SQL Server Query Optimizer выбирает лучший индекс, подходящий для запроса, но из-за отсутствия / устаревших статистических данных или особых потребностей вы можете заставить его.

```
SELECT *
FROM mytable WITH (INDEX (ix_date))
WHERE field1 > 0
    AND CreationDate > '20170101'
```

Прочитайте Подсказки по запросам онлайн: https://riptutorial.com/ru/sql-server/topic/6881/подсказки-по-запросам

глава 69: Полнотекстовая индексация

Examples

A. Создание уникального индекса, полнотекстового каталога и полнотекстового индекса

В следующем примере создается уникальный индекс в столбце JobCandidateID таблицы HumanResources.JobCandidate базы данных примеров AdventureWorks2012. Затем пример создает полный текстовый каталог по умолчанию, ft. Наконец, пример создает полнотекстовый индекс в столбце Resume, используя каталог ft и системный стоп-лист.

```
USE AdventureWorks2012;
GO
CREATE UNIQUE INDEX ui_ukJobCand ON HumanResources.JobCandidate(JobCandidateID);
CREATE FULLTEXT CATALOG ft AS DEFAULT;
CREATE FULLTEXT INDEX ON HumanResources.JobCandidate(Resume)
   KEY INDEX ui_ukJobCand
   WITH STOPLIST = SYSTEM;
GO
```

https://www.simple-talk.com/sql/learn-sql-server/understanding-full-text-indexing-in-sql-server/

https://msdn.microsoft.com/en-us/library/cc879306.aspx

https://msdn.microsoft.com/en-us/library/ms142571.aspx

Создание полнотекстового индекса в нескольких столбцах таблицы

```
USE AdventureWorks2012;
GO
CREATE FULLTEXT CATALOG production_catalog;
GO
CREATE FULLTEXT INDEX ON Production.ProductReview
 (
  ReviewerName
     Language 1033,
  EmailAddress
     Language 1033,
  Comments
     Language 1033
 )
  KEY INDEX PK_ProductReview_ProductReviewID
      ON production_catalog;
GO
```

Создание полнотекстового индекса со списком свойств поиска без его заполнения

```
USE AdventureWorks2012;
GO
CREATE FULLTEXT INDEX ON Production.Document
  (
  Title
      Language 1033,
  DocumentSummary
      Language 1033,
  Document
      TYPE COLUMN FileExtension
      Language 1033
  )
  KEY INDEX PK_Document_DocumentID
         WITH STOPLIST = SYSTEM, SEARCH PROPERTY LIST = DocumentPropertyList, CHANGE_TRACKING
OFF, NO POPULATION;
   GO
```

И заселение его позже

```
ALTER FULLTEXT INDEX ON Production.Document SET CHANGE_TRACKING AUTO;
GO
```

Полнотекстовый поиск

```
SELECT product_id
FROM products
WHERE CONTAINS(product_description, "Snap Happy 100EZ" OR FORMSOF(THESAURUS,'Snap Happy') OR
'100EZ')
AND product_cost < 200 ;


SELECT candidate_name,SSN
FROM candidates
WHERE CONTAINS(candidate_resume,"SQL Server") AND candidate_division =DBA;
```

Для получения более подробной информации https://msdn.microsoft.com/en-us/library/ms142571.aspx

Прочитайте Полнотекстовая индексация онлайн: https://riptutorial.com/ru/sql-server/topic/4557/полнотекстовая-индексация

глава 70: Получить информацию о базе данных

замечания

Как и в других системах реляционных баз данных, SQL Server предоставляет метаданные о ваших базах данных.

Это обеспечивается через стандартную схему ISO `INFORMATION_SCHEMA` или представления каталога `sys` каталогов SQL Server.

Examples

Подсчитать количество таблиц в базе данных

Этот запрос вернет число таблиц в указанной базе данных.

```
USE YourDatabaseName
SELECT COUNT(*) from INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE'
```

Следующий способ можно сделать для всех пользовательских таблиц с SQL Server 2008+. Ссылка здесь .

```
SELECT COUNT(*) FROM sys.tables
```

Получить список всех сохраненных процедур

Следующие запросы возвращают список всех хранимых процедур в базе данных с базовой информацией о каждой сохраненной процедуре:

SQL Server 2005

```
SELECT *
FROM INFORMATION_SCHEMA.ROUTINES
WHERE ROUTINE_TYPE = 'PROCEDURE'
```

Наиболее `ROUTINE_NAME` `ROUTINE_SCHEMA` `ROUTINE_NAME` , `ROUTINE_SCHEMA` и `ROUTINE_DEFINITION` .

SQL Server 2005

```
SELECT *
FROM sys.objects
WHERE type = 'P'
```

SQL Server 2005

```
SELECT *
FROM sys.procedures
```

Обратите внимание, что эта версия имеет преимущество перед выбором из sys.objects, поскольку она включает дополнительные столбцы `is_auto_executed` , `is_execution_replicated` , `is_repl_serializable` И `skips_repl_constraints` .

SQL Server 2005

```
SELECT *
FROM sysobjects
WHERE type = 'P'
```

Обратите внимание, что вывод содержит много столбцов, которые никогда не будут связаны с хранимой процедурой.

Следующий набор запросов вернет все хранимые процедуры в базе данных, содержащие строку «SearchTerm»:

SQL Server 2005

```
SELECT o.name
FROM syscomments c
INNER JOIN sysobjects o
    ON c.id=o.id
WHERE o.xtype = 'P'
    AND c.TEXT LIKE '%SearchTerm%'
```

SQL Server 2005

```
SELECT p.name
FROM sys.sql_modules AS m
INNER JOIN sys.procedures AS p
    ON m.object_id = p.object_id
WHERE definition LIKE '%SearchTerm%'
```

Получить список всех баз данных на сервере

Метод 1: ниже запрос будет применим для версии SQL Server 2000+ (содержит 12 столбцов)

```
SELECT * FROM dbo.sysdatabases
```

Метод 2: ниже запроса извлекайте информацию о базах данных с большей информацией (например, состояние, изоляция, модель восстановления и т. Д.).

Примечание. Это представление каталога и будут доступны версии SQL SERVER 2005+

```
SELECT * FROM sys.databases
```

Способ 3. Чтобы увидеть только имена баз данных, вы можете использовать недокументированные sp_MSForEachDB

```
EXEC sp_MSForEachDB 'SELECT ''?'' AS DatabaseName'
```

Способ 4. Ниже SP поможет вам указать размер базы данных вместе с именем базы данных, владельцем, статусом и т. Д. На сервере

```
EXEC sp_helpdb
```

Метод 5 Аналогично, ниже хранимой процедуры будут указаны имя базы данных, размер базы данных и примечания

```
EXEC sp_databases
```

Файлы базы данных

Отображение всех файлов данных для всех баз данных с информацией о размере и увеличении

```
SELECT  d.name AS 'Database',
        d.database_id,
        SF.fileid,
        SF.name AS 'LogicalFileName',
        CASE SF.status & 0x100000
            WHEN 1048576 THEN 'Percentage'
            WHEN 0 THEN 'MB'
        END AS 'FileGrowthOption',
        Growth AS GrowthUnit,
        ROUND(((CAST(Size AS FLOAT)*8)/1024)/1024,2) [SizeGB], -- Convert 8k pages to GB
        Maxsize,
        filename AS PhysicalFileName

FROM    Master.SYS.SYSALTFILES SF
Join    Master.SYS.Databases d on sf.fileid = d.database_id

Order by d.name
```

Извлечь параметры базы данных

Следующий запрос возвращает параметры базы данных и метаданные:

```
select * from sys.databases WHERE name = 'MyDatabaseName';
```

Показать размер всех таблиц в текущей базе данных

```
SELECT
    s.name + '.' + t.NAME AS TableName,
    SUM(a.used_pages)*8 AS 'TableSizeKB'  --a page in SQL Server is 8kb
```

```
FROM sys.tables t
    JOIN sys.schemas s on t.schema_id = s.schema_id
    LEFT JOIN sys.indexes i ON t.OBJECT_ID = i.object_id
    LEFT JOIN sys.partitions p ON i.object_id = p.OBJECT_ID AND i.index_id = p.index_id
    LEFT JOIN sys.allocation_units a ON p.partition_id = a.container_id
GROUP BY
    s.name, t.name
ORDER BY
    --Either sort by name:
    s.name + '.' + t.NAME
    --Or sort largest to smallest:
    --SUM(a.used_pages) desc
```

Определение пути разрешения входа в Windows

Это покажет тип пользователя и путь разрешения (из которых Windows-группа получает свои разрешения).

```
xp_logininfo 'DOMAIN\user'
```

Извлечение таблиц, содержащих известную колонку

Этот запрос вернет все `COLUMNS` и связанные с ними `TABLES` для данного имени столбца. Он предназначен, чтобы показать вам, какие таблицы (неизвестные) содержат указанный столбец (известный)

```
SELECT
    c.name AS ColName,
    t.name AS TableName
FROM
    sys.columns c
    JOIN sys.tables t ON c.object_id = t.object_id
WHERE
    c.name LIKE '%MyName%'
```

Посмотрите, используются ли функции, специфичные для предприятия

Иногда бывает полезно проверить, что ваша работа над выпуском Developer не ввела зависимости от любых функций, ограниченных Enterprise edition.

Вы можете сделать это, используя системный вид `sys.dm_db_persisted_sku_features` , например:

```
SELECT * FROM sys.dm_db_persisted_sku_features
```

Против самой базы данных.

При этом будут перечислены используемые функции, если они есть.

Поиск и возврат всех таблиц и столбцов, содержащих заданное значение столбца

Этот скрипт, здесь и здесь , вернет все таблицы и столбцы, где существует указанное значение. Это очень важно для определения того, где определенное значение находится в базе данных. Он может облагаться налогом, поэтому предлагается сначала выполнить его в резервной / тестовой среде.

```
DECLARE @SearchStr nvarchar(100)
SET @SearchStr = '## YOUR STRING HERE ##'


    -- Copyright © 2002 Narayana Vyas Kondreddi. All rights reserved.
    -- Purpose: To search all columns of all tables for a given search string
    -- Written by: Narayana Vyas Kondreddi
    -- Site: http://vyaskn.tripod.com
    -- Updated and tested by Tim Gaunt
    -- http://www.thesitedoctor.co.uk
    --
http://blogs.thesitedoctor.co.uk/tim/2010/02/19/Search+Every+Table+And+Field+In+A+SQL+Server+Database+U

    -- Tested on: SQL Server 7.0, SQL Server 2000, SQL Server 2005 and SQL Server 2010
    -- Date modified: 03rd March 2011 19:00 GMT
    CREATE TABLE #Results (ColumnName nvarchar(370), ColumnValue nvarchar(3630))

    SET NOCOUNT ON

    DECLARE @TableName nvarchar(256), @ColumnName nvarchar(128), @SearchStr2 nvarchar(110)
    SET  @TableName = ''
    SET @SearchStr2 = QUOTENAME('%' + @SearchStr + '%','''')

    WHILE @TableName IS NOT NULL

    BEGIN
        SET @ColumnName = ''
        SET @TableName =
        (
            SELECT MIN(QUOTENAME(TABLE_SCHEMA) + '.' + QUOTENAME(TABLE_NAME))
            FROM     INFORMATION_SCHEMA.TABLES
            WHERE         TABLE_TYPE = 'BASE TABLE'
                AND    QUOTENAME(TABLE_SCHEMA) + '.' + QUOTENAME(TABLE_NAME) > @TableName
                AND    OBJECTPROPERTY(
                        OBJECT_ID(
                            QUOTENAME(TABLE_SCHEMA) + '.' + QUOTENAME(TABLE_NAME)
                             ), 'IsMSShipped'
                               ) = 0
        )

        WHILE (@TableName IS NOT NULL) AND (@ColumnName IS NOT NULL)

        BEGIN
            SET @ColumnName =
            (
                SELECT MIN(QUOTENAME(COLUMN_NAME))
                FROM     INFORMATION_SCHEMA.COLUMNS
                WHERE         TABLE_SCHEMA    = PARSENAME(@TableName, 2)
                    AND    TABLE_NAME    = PARSENAME(@TableName, 1)
                    AND    DATA_TYPE IN ('char', 'varchar', 'nchar', 'nvarchar', 'int',
```

```
'decimal')
                    AND    QUOTENAME(COLUMN_NAME) > @ColumnName
            )

            IF @ColumnName IS NOT NULL

            BEGIN
                INSERT INTO #Results
                EXEC
                (
                    'SELECT ''' + @TableName + '.' + @ColumnName + ''', LEFT(' + @ColumnName +
', 3630) FROM ' + @TableName + ' (NOLOCK) ' +
                    ' WHERE ' + @ColumnName + ' LIKE ' + @SearchStr2
                )
            END
        END
    END

    SELECT ColumnName, ColumnValue FROM #Results

DROP TABLE #Results
- See more at: http://thesitedoctor.co.uk/blog/search-every-table-and-field-in-a-sql-server-
database-updated#sthash.bBEqfJVZ.dpuf
```

Получить все схемы, таблицы, столбцы и индексы

```
SELECT
    s.name AS [schema],
    t.object_id AS [table_object_id],
    t.name AS [table_name],
    c.column_id,
    c.name AS [column_name],
    i.name AS [index_name],
    i.type_desc AS [index_type]
FROM sys.schemas AS s
INNER JOIN sys.tables AS t
    ON s.schema_id = t.schema_id
INNER JOIN sys.columns AS c
    ON t.object_id = c.object_id
LEFT JOIN sys.index_columns AS ic
    ON c.object_id = ic.object_id and c.column_id = ic.column_id
LEFT JOIN sys.indexes AS i
    ON ic.object_id = i.object_id and ic.index_id = i.index_id
ORDER BY [schema], [table_name], c.column_id;
```

Вернуть список заданий агента SQL с информацией о расписании

```
USE msdb
Go


SELECT dbo.sysjobs.Name AS 'Job Name',
      'Job Enabled' = CASE dbo.sysjobs.Enabled
            WHEN 1 THEN 'Yes'
            WHEN 0 THEN 'No'
      END,
      'Frequency' = CASE dbo.sysschedules.freq_type
```

```
        WHEN 1 THEN 'Once'
        WHEN 4 THEN 'Daily'
        WHEN 8 THEN 'Weekly'
        WHEN 16 THEN 'Monthly'
        WHEN 32 THEN 'Monthly relative'
        WHEN 64 THEN 'When SQLServer Agent starts'
    END,
    'Start Date' = CASE active_start_date
        WHEN 0 THEN null
        ELSE
        substring(convert(varchar(15),active_start_date),1,4) + '/' +
        substring(convert(varchar(15),active_start_date),5,2) + '/' +
        substring(convert(varchar(15),active_start_date),7,2)
    END,
    'Start Time' = CASE len(active_start_time)
        WHEN 1 THEN cast('00:00:0' + right(active_start_time,2) as char(8))
        WHEN 2 THEN cast('00:00:' + right(active_start_time,2) as char(8))
        WHEN 3 THEN cast('00:0'
                    + Left(right(active_start_time,3),1)
                    +':' + right(active_start_time,2) as char (8))
        WHEN 4 THEN cast('00:'
                    + Left(right(active_start_time,4),2)
                    +':' + right(active_start_time,2) as char (8))
        WHEN 5 THEN cast('0'
                    + Left(right(active_start_time,5),1)
                    +':' + Left(right(active_start_time,4),2)
                    +':' + right(active_start_time,2) as char (8))
        WHEN 6 THEN cast(Left(right(active_start_time,6),2)
                    +':' + Left(right(active_start_time,4),2)
                    +':' + right(active_start_time,2) as char (8))
    END,

    CASE len(run_duration)
        WHEN 1 THEN cast('00:00:0'
                    + cast(run_duration as char) as char (8))
        WHEN 2 THEN cast('00:00:'
                    + cast(run_duration as char) as char (8))
        WHEN 3 THEN cast('00:0'
                    + Left(right(run_duration,3),1)
                    +':' + right(run_duration,2) as char (8))
        WHEN 4 THEN cast('00:'
                    + Left(right(run_duration,4),2)
                    +':' + right(run_duration,2) as char (8))
        WHEN 5 THEN cast('0'
                    + Left(right(run_duration,5),1)
                    +':' + Left(right(run_duration,4),2)
                    +':' + right(run_duration,2) as char (8))
        WHEN 6 THEN cast(Left(right(run_duration,6),2)
                    +':' + Left(right(run_duration,4),2)
                    +':' + right(run_duration,2) as char (8))
    END as 'Max Duration',
  CASE(dbo.sysschedules.freq_subday_interval)
        WHEN 0 THEN 'Once'
        ELSE cast('Every '
                    + right(dbo.sysschedules.freq_subday_interval,2)
                    + ' '
                    +     CASE(dbo.sysschedules.freq_subday_type)
                                WHEN 1 THEN 'Once'
                                WHEN 4 THEN 'Minutes'
                                WHEN 8 THEN 'Hours'
                            END as char(16))
```

```
    END as 'Subday Frequency'
FROM dbo.sysjobs
LEFT OUTER JOIN dbo.sysjobschedules
ON dbo.sysjobs.job_id = dbo.sysjobschedules.job_id
INNER JOIN dbo.sysschedules ON dbo.sysjobschedules.schedule_id = dbo.sysschedules.schedule_id
LEFT OUTER JOIN (SELECT job_id, max(run_duration) AS run_duration
            FROM dbo.sysjobhistory
            GROUP BY job_id) Q1
ON dbo.sysjobs.job_id = Q1.job_id
WHERE Next_run_time = 0

UNION

SELECT dbo.sysjobs.Name AS 'Job Name',
      'Job Enabled' = CASE dbo.sysjobs.Enabled
            WHEN 1 THEN 'Yes'
            WHEN 0 THEN 'No'
      END,
      'Frequency' = CASE dbo.sysschedules.freq_type
            WHEN 1 THEN 'Once'
            WHEN 4 THEN 'Daily'
            WHEN 8 THEN 'Weekly'
            WHEN 16 THEN 'Monthly'
            WHEN 32 THEN 'Monthly relative'
            WHEN 64 THEN 'When SQLServer Agent starts'
      END,
      'Start Date' = CASE next_run_date
            WHEN 0 THEN null
            ELSE
            substring(convert(varchar(15),next_run_date),1,4) + '/' +
            substring(convert(varchar(15),next_run_date),5,2) + '/' +
            substring(convert(varchar(15),next_run_date),7,2)
      END,
      'Start Time' = CASE len(next_run_time)
            WHEN 1 THEN cast('00:00:0' + right(next_run_time,2) as char(8))
            WHEN 2 THEN cast('00:00:' + right(next_run_time,2) as char(8))
            WHEN 3 THEN cast('00:0'
                        + Left(right(next_run_time,3),1)
                        +':' + right(next_run_time,2) as char (8))
            WHEN 4 THEN cast('00:'
                        + Left(right(next_run_time,4),2)
                        +':' + right(next_run_time,2) as char (8))
            WHEN 5 THEN cast('0' + Left(right(next_run_time,5),1)
                        +':' + Left(right(next_run_time,4),2)
                        +':' + right(next_run_time,2) as char (8))
            WHEN 6 THEN cast(Left(right(next_run_time,6),2)
                        +':' + Left(right(next_run_time,4),2)
                        +':' + right(next_run_time,2) as char (8))
      END,

      CASE len(run_duration)
            WHEN 1 THEN cast('00:00:0'
                        + cast(run_duration as char) as char (8))
            WHEN 2 THEN cast('00:00:'
                        + cast(run_duration as char) as char (8))
            WHEN 3 THEN cast('00:0'
                        + Left(right(run_duration,3),1)
                        +':' + right(run_duration,2) as char (8))
            WHEN 4 THEN cast('00:'
                        + Left(right(run_duration,4),2)
                        +':' + right(run_duration,2) as char (8))
```

```
            WHEN 5 THEN cast('0'
                        + Left(right(run_duration,5),1)
                        +':' + Left(right(run_duration,4),2)
                        +':' + right(run_duration,2) as char (8))
            WHEN 6 THEN cast(Left(right(run_duration,6),2)
                        +':' + Left(right(run_duration,4),2)
                        +':' + right(run_duration,2) as char (8))
      END as 'Max Duration',
    CASE(dbo.sysschedules.freq_subday_interval)
            WHEN 0 THEN 'Once'
            ELSE cast('Every '
                        + right(dbo.sysschedules.freq_subday_interval,2)
                        + ' '
                        +     CASE(dbo.sysschedules.freq_subday_type)
                                    WHEN 1 THEN 'Once'
                                    WHEN 4 THEN 'Minutes'
                                    WHEN 8 THEN 'Hours'
                                END as char(16))
    END as 'Subday Frequency'
FROM dbo.sysjobs
LEFT OUTER JOIN dbo.sysjobschedules ON dbo.sysjobs.job_id = dbo.sysjobschedules.job_id
INNER JOIN dbo.sysschedules ON dbo.sysjobschedules.schedule_id = dbo.sysschedules.schedule_id
LEFT OUTER JOIN (SELECT job_id, max(run_duration) AS run_duration
            FROM dbo.sysjobhistory
            GROUP BY job_id) Q1
ON dbo.sysjobs.job_id = Q1.job_id
WHERE Next_run_time <> 0

ORDER BY [Start Date],[Start Time]
```

Получение информации о действиях резервного копирования и восстановления

Чтобы получить список всех операций резервного копирования, выполняемых в текущем экземпляре базы данных:

```
SELECT sdb.Name AS DatabaseName,
    COALESCE(CONVERT(VARCHAR(50), bus.backup_finish_date, 120),'-') AS LastBackUpDateTime
FROM sys.sysdatabases sdb
    LEFT OUTER JOIN msdb.dbo.backupset bus ON bus.database_name = sdb.name
ORDER BY sdb.name, bus.backup_finish_date DESC
```

Чтобы получить список всех операций восстановления, выполненных в текущем экземпляре базы данных:

```
SELECT
    [d].[name] AS database_name,
    [r].restore_date AS last_restore_date,
    [r].[user_name],
    [bs].[backup_finish_date] AS backup_creation_date,
    [bmf].[physical_device_name] AS [backup_file_used_for_restore]
FROM master.sys.databases [d]
    LEFT OUTER JOIN msdb.dbo.[restorehistory] r ON r.[destination_database_name] = d.Name
    INNER JOIN msdb.dbo.backupset [bs] ON [r].[backup_set_id] = [bs].[backup_set_id]
    INNER JOIN msdb.dbo.backupmediafamily bmf ON [bs].[media_set_id] = [bmf].[media_set_id]
ORDER BY [d].[name], [r].restore_date DESC
```

Найти каждое упоминание поля в базе данных

```
SELECT DISTINCT
 o.name AS Object_Name,o.type_desc
 FROM sys.sql_modules m
    INNER JOIN sys.objects  o ON m.object_id=o.object_id
 WHERE m.definition Like '%myField%'
 ORDER BY 2,1
```

`myField` упоминания `myField` в SProcs, Views и т. Д.

Прочитайте Получить информацию о базе данных онлайн: https://riptutorial.com/ru/sql-server/topic/697/получить-информацию-о-базе-данных

глава 71: Получить информацию о вашем экземпляре

Examples

Получение локальных и удаленных серверов

Чтобы получить список всех серверов, зарегистрированных в экземпляре:

```
EXEC sp_helpserver;
```

Получить информацию о текущих сеансах и выполнении запросов

```
sp_who2
```

Эта процедура может использоваться для поиска информации о текущих сеансах SQL-сервера. Поскольку это процедура, часто полезно сохранять результаты во временную таблицу или переменную таблицы, чтобы можно было заказывать, фильтровать и преобразовывать результаты по мере необходимости.

Нижеследующее может использоваться для запрашиваемой версии `sp_who2` :

```
-- Create a variable table to hold the results of sp_who2 for querying purposes

DECLARE @who2 TABLE (
      SPID INT NULL,
      Status VARCHAR(1000) NULL,
      Login SYSNAME NULL,
      HostName SYSNAME NULL,
      BlkBy SYSNAME NULL,
      DBName SYSNAME NULL,
      Command VARCHAR(8000) NULL,
      CPUTime INT NULL,
      DiskIO INT NULL,
      LastBatch VARCHAR(250) NULL,
      ProgramName VARCHAR(250) NULL,
      SPID2 INT NULL, -- a second SPID for some reason...?
      REQUESTID INT NULL
)

INSERT INTO @who2
EXEC sp_who2

SELECT    *
FROM    @who2 w
WHERE    1=1
```

Примеры:

```
-- Find specific user sessions:
SELECT  *
FROM    @who2 w
WHERE   1=1
    and  login = 'userName'

-- Find longest CPUTime queries:
SELECT  top 5 *
FROM    @who2 w
WHERE   1=1
order   by CPUTime desc
```

Восстановить выпуск и версию экземпляра

```
SELECT    SERVERPROPERTY('ProductVersion') AS ProductVersion,
          SERVERPROPERTY('ProductLevel') AS ProductLevel,
          SERVERPROPERTY('Edition') AS Edition,
          SERVERPROPERTY('EngineEdition') AS EngineEdition;
```

Получить время ожидания экземпляра в днях

```
SELECT  DATEDIFF(DAY, login_time, getdate()) UpDays
FROM    master..sysprocesses
WHERE   spid = 1
```

Информация о версии SQL Server

Чтобы узнать выпуск SQL Server, уровень продукта и номер версии, а также имя хост-компьютера и тип сервера:

```
SELECT    SERVERPROPERTY('MachineName') AS Host,
          SERVERPROPERTY('InstanceName') AS Instance,
          DB_NAME() AS DatabaseContext,
          SERVERPROPERTY('Edition') AS Edition,
          SERVERPROPERTY('ProductLevel') AS ProductLevel,
          CASE SERVERPROPERTY('IsClustered')
            WHEN 1 THEN 'CLUSTERED'
            ELSE 'STANDALONE' END AS ServerType,
          @@VERSION AS VersionNumber;
```

Общая информация о базах данных, таблицах, хранимых процедурах и порядке их поиска.

Запрос на поиск последних выполненных sp's в db

```
SELECT execquery.last_execution_time AS [Date Time], execsql.text AS [Script]
FROM sys.dm_exec_query_stats AS execquery
CROSS APPLY sys.dm_exec_sql_text(execquery.sql_handle) AS execsql
ORDER BY execquery.last_execution_time DESC
```

Запрос на поиск по хранимым процедурам

```
SELECT o.type_desc AS ROUTINE_TYPE,o.[name] AS ROUTINE_NAME,
m.definition AS ROUTINE_DEFINITION
FROM sys.sql_modules AS m INNER JOIN sys.objects AS o
ON m.object_id = o.object_id WHERE m.definition LIKE '%Keyword%'
order by ROUTINE_NAME
```

Запрос на поиск столбца из всех таблиц базы данных

```
SELECT t.name AS table_name,
SCHEMA_NAME(schema_id) AS schema_name,
c.name AS column_name
FROM sys.tables AS t
INNER JOIN sys.columns c ON t.OBJECT_ID = c.OBJECT_ID
where c.name like 'Keyword%'
ORDER BY schema_name, table_name;
```

Запрос на восстановление данных восстановления

```
WITH LastRestores AS
(
SELECT
    DatabaseName = [d].[name] ,
    [d].[create_date] ,
    [d].[compatibility_level] ,
    [d].[collation_name] ,
    r.*,
    RowNum = ROW_NUMBER() OVER (PARTITION BY d.Name ORDER BY r.[restore_date] DESC)
FROM master.sys.databases d
LEFT OUTER JOIN msdb.dbo.[restorehistory] r ON r.[destination_database_name] = d.Name
)
SELECT *
FROM [LastRestores]
WHERE [RowNum] = 1
```

Запрос на поиск журнала

```
select top 100 * from databaselog
Order by Posttime desc
```

Запросить информацию о Sps

```
SELECT name, create_date, modify_date
FROM sys.objects
WHERE type = 'P'
Order by modify_date desc
```

Прочитайте Получить информацию о вашем экземпляре онлайн: https://riptutorial.com/ru/sql-server/topic/2029/получить-информацию-о-вашем-экземпляре

глава 72: ПОПРОБУЙ ПОЙМАТЬ

замечания

TRY / CATCH - это языковая конструкция, специфичная для T-SQL MS SQL Server.

Он позволяет обрабатывать ошибки в T-SQL, подобно тому, как это видно в коде .NET.

Examples

Транзакция в TRY / CATCH

Это приведет к откату обеих вставок из-за недопустимого datetime:

```
BEGIN TRANSACTION
BEGIN TRY
    INSERT INTO dbo.Sale(Price, SaleDate, Quantity)
    VALUES (5.2, GETDATE(), 1)
    INSERT INTO dbo.Sale(Price, SaleDate, Quantity)
    VALUES (5.2, 'not a date', 1)
    COMMIT TRANSACTION
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION -- First Rollback and then throw.
    THROW
END CATCH
```

Это зафиксирует обе вставки:

```
BEGIN TRANSACTION
BEGIN TRY
    INSERT INTO dbo.Sale(Price, SaleDate, Quantity)
    VALUES (5.2, GETDATE(), 1)
    INSERT INTO dbo.Sale(Price, SaleDate, Quantity)
    VALUES (5.2, GETDATE(), 1)
    COMMIT TRANSACTION
END TRY
BEGIN CATCH
    THROW
    ROLLBACK TRANSACTION
END CATCH
```

Повышение ошибок в блоке try-catch

Функция RAISERROR генерирует ошибку в блоке TRY CATCH:

```
DECLARE @msg nvarchar(50) = 'Here is a problem!'
BEGIN TRY
    print 'First statement';
```

```
    RAISERROR(@msg, 11, 1);
    print 'Second statement';
END TRY
BEGIN CATCH
    print 'Error: ' + ERROR_MESSAGE();
END CATCH
```

RAISERROR со вторым параметром больше 10 (11 в этом примере) прекратит выполнение в TRY BLOCK и вызовет ошибку, которая будет обрабатываться в блоке CATCH. Вы можете получить доступ к сообщению об ошибке, используя функцию ERROR_MESSAGE (). Выход этого образца:

```
First statement
Error: Here is a problem!
```

Получение информационных сообщений в блоке try catch

RAISERROR с серьезностью (второй параметр), меньшей или равной 10, не будет генерировать исключение.

```
BEGIN TRY
    print 'First statement';
    RAISERROR( 'Here is a problem!', 10, 15);
    print 'Second statement';
END TRY
BEGIN CATCH
    print 'Error: ' + ERROR_MESSAGE();
END CATCH
```

После утверждения RAISERROR будет выполнен третий оператор, и блок CATCH не будет вызываться. Результатом выполнения является:

```
First statement
Here is a problem!
Second statement
```

Исключение переброски, созданное RAISERROR

Вы можете повторно выбросить ошибку, которую вы поймаете в блоке CATCH, используя инструкцию TRHOW:

```
DECLARE @msg nvarchar(50) = 'Here is a problem! Area: ''%s'' Line:''%i'''
BEGIN TRY
    print 'First statement';
    RAISERROR(@msg, 11, 1, 'TRY BLOCK', 2);
    print 'Second statement';
END TRY
BEGIN CATCH
    print 'Error: ' + ERROR_MESSAGE();
    THROW;
END CATCH
```

Обратите внимание, что в этом случае мы поднимаем ошибку с форматированными аргументами (четвертый и пятый параметры). Это может быть полезно, если вы хотите добавить дополнительную информацию в сообщение. Результатом выполнения является:

```
First statement
Error: Here is a problem! Area: 'TRY BLOCK' Line:'2'
Msg 50000, Level 11, State 1, Line 26
Here is a problem! Area: 'TRY BLOCK' Line:'2'
```

Исключение исключения в блоках TRY / CATCH

Вы можете создать исключение в блоке try catch:

```
DECLARE @msg nvarchar(50) = 'Here is a problem!'
BEGIN TRY
    print 'First statement';
    THROW 51000, @msg, 15;
    print 'Second statement';
END TRY
BEGIN CATCH
    print 'Error: ' + ERROR_MESSAGE();
    THROW;
END CATCH
```

Исключение с обработкой в блоке CATCH, а затем повторно выбрасывается с использованием THROW без параметров.

```
First statement
Error: Here is a problem!
Msg 51000, Level 16, State 15, Line 39
Here is a problem!
```

THROW похож на RAISERROR со следующими отличиями:

- Рекомендация заключается в том, что в новых приложениях вместо RASIERROR следует использовать THROW.
- THROW может использовать любое число в качестве первого аргумента (номер ошибки), RAISERROR может использовать только идентификаторы в представлении sys.messages
- THROW имеет серьезность 16 (нельзя изменить)
- THROW не может форматировать аргументы, такие как RAISERROR. Используйте функцию FORMATMESSAGE в качестве аргумента RAISERROR, если вам нужна эта функция.

Прочитайте ПОПРОБУЙ ПОЙМАТЬ онлайн: https://riptutorial.com/ru/sql-server/topic/5189/попробуй-поймать

глава 73: Последняя введенная идентификация

Examples

SCOPE_IDENTITY ()

```
CREATE TABLE dbo.logging_table(log_id INT IDENTITY(1,1) PRIMARY KEY,
                               log_message VARCHAR(255))

CREATE TABLE dbo.person(person_id INT IDENTITY(1,1) PRIMARY KEY,
                        person_name VARCHAR(100) NOT NULL)
GO;

CREATE TRIGGER dbo.InsertToADifferentTable ON dbo.person
AFTER INSERT
AS
    INSERT INTO dbo.logging_table(log_message)
    VALUES('Someone added something to the person table')
GO;

INSERT INTO dbo.person(person_name)
VALUES('John Doe')

SELECT SCOPE_IDENTITY();
```

Это вернет последнее добавленное значение идентичности, созданное в том же соединении, в пределах текущей области. В этом случае 1, для первой строки в таблице dbo.person.

@@ IDENTITY

```
CREATE TABLE dbo.logging_table(log_id INT IDENTITY(1,1) PRIMARY KEY,
                               log_message VARCHAR(255))

CREATE TABLE dbo.person(person_id INT IDENTITY(1,1) PRIMARY KEY,
                        person_name VARCHAR(100) NOT NULL)
GO;

CREATE TRIGGER dbo.InsertToADifferentTable ON dbo.person
AFTER INSERT
AS
    INSERT INTO dbo.logging_table(log_message)
    VALUES('Someone added something to the person table')
GO;

INSERT INTO dbo.person(person_name)
VALUES('John Doe')

SELECT @@IDENTITY;
```

Это вернет самую последнюю добавленную идентификацию в том же соединении, независимо от области действия. В этом случае, каково бы ни было текущее значение столбца идентификации в logging_table, при условии, что никакая другая активность не возникает на экземпляре SQL Server, и никакие другие триггеры не запускают эту вставку.

IDENT_CURRENT ('имя_таблицы')

```
SELECT IDENT_CURRENT('dbo.person');
```

Это позволит выбрать последнее добавленное значение идентичности в выбранной таблице, независимо от того, какое соединение или область действия.

@@ IDENTITY и MAX (ID)

```
SELECT MAX(Id) FROM Employees -- Display the value of Id in the last row in Employees table.
GO
INSERT INTO Employees (FName, LName, PhoneNumber) -- Insert a new row
VALUES ('John', 'Smith', '25558696525')
GO
SELECT @@IDENTITY
GO
SELECT MAX(Id) FROM Employees -- Display the value of Id of the newly inserted row.
GO
```

Последние два значения операторов SELECT совпадают.

Прочитайте Последняя введенная идентификация онлайн: https://riptutorial.com/ru/sql-server/topic/5674/последняя-введенная-идентификация

глава 74: Последовательности

Examples

Создать последовательность

```
CREATE SEQUENCE [dbo].[CustomersSeq]
AS INT
START WITH 10001
INCREMENT BY 1
MINVALUE -1;
```

Использовать последовательность в таблице

```
CREATE TABLE [dbo].[Customers]
(
    CustomerID INT DEFAULT (NEXT VALUE FOR [dbo].[CustomersSeq]) NOT NULL,
    CustomerName VARCHAR(100),
);
```

Вставить в таблицу с последовательностью

```
INSERT INTO [dbo].[Customers]
       ([CustomerName])
 VALUES
       ('Jerry'),
       ('Gorge')

SELECT * FROM [dbo].[Customers]
```

Результаты

Пользовательский ИД	Имя покупателя
10001	Джерри
10002	теснина

Удалить и добавить новый

```
DELETE FROM [dbo].[Customers]
WHERE CustomerName = 'Gorge';

INSERT INTO [dbo].[Customers]
       ([CustomerName])
 VALUES ('George')
```

```
SELECT * FROM [dbo].[Customers]
```

Результаты

Пользовательский ИД	Имя покупателя
10001	Джерри
10003	Джордж

Прочитайте Последовательности онлайн: https://riptutorial.com/ru/sql-server/topic/5324/последовательности

глава 75: Преобразование типов данных

Examples

TRY PARSE

SQL Server 2012

Он преобразует строковый тип данных в целевой тип данных (Date или Numeric).

Например, исходные данные являются строковыми типами, и нам нужно скрывать тип даты. Если попытка конверсии не удалась, она возвращает значение NULL.

Синтаксис: TRY_PARSE (string_value AS data_type [ИСПОЛЬЗОВАНИЕ культуры])

String_value - аргумент является значением источника, которое является типом NVARCHAR (4000).
Data_type - этот аргумент является типом данных цели или даты или числа.
Культура. Это необязательный аргумент, который помогает преобразовать значение в формат Culture. Предположим, вы хотите отобразить дату на французском языке, тогда вам нужно передать тип культуры как «Fr-FR». Если вы не пройдете какое-либо действительное название культуры, PARSE вызовет ошибку.

```
DECLARE @fakeDate AS varchar(10);
DECLARE @realDate AS VARCHAR(10);
SET @fakeDate = 'iamnotadate';
SET @realDate = '13/09/2015';

SELECT TRY_PARSE(@fakeDate AS DATE); --NULL  as the parsing fails

SELECT TRY_PARSE(@realDate AS DATE); -- NULL due to type mismatch

SELECT TRY_PARSE(@realDate AS DATE USING 'Fr-FR'); -- 2015-09-13
```

TRY CONVERT

SQL Server 2012

Он преобразует значение в указанный тип данных, и если преобразование не выполняется, оно возвращает NULL. Например, исходное значение в формате строки, и нам нужен формат даты / целого. Тогда это поможет нам достичь того же.

Синтаксис: TRY_CONVERT (data_type [(длина)], выражение [, style])

TRY_CONVERT () возвращает значение, присвоенное указанному типу данных, если преследование преуспевает; в противном случае возвращает null.

Data_type - тип данных, в который нужно преобразовать. Здесь length - необязательный параметр, который помогает получить результат в указанной длине.
Выражение - значение, которое нужно преобразовать
Стиль. Это необязательный параметр, который определяет форматирование.
Предположим, вам нужен формат даты, например «18 мая 2013 года», тогда вам нужно пройти стиль «111».

```
DECLARE @sampletext AS VARCHAR(10);
SET @sampletext = '123456';
DECLARE @ realDate AS VARCHAR(10);
SET @realDate = '13/09/2015’;
SELECT TRY_CONVERT(INT, @sampletext); -- 123456
SELECT TRY_CONVERT(DATETIME, @sampletext); -- NULL
SELECT TRY_CONVERT(DATETIME, @realDate, 111); -- Sep, 13 2015
```

ПОПРОБУЙТЕ CAST

SQL Server 2012

Он преобразует значение в указанный тип данных, и если преобразование не выполняется, оно возвращает NULL. Например, исходное значение в формате строки, и нам нужно его в формате double / integer. Тогда это поможет нам в достижении этого.

Синтаксис: TRY_CAST (выражение AS data_type [(длина)])

TRY_CAST () возвращает значение, отлитое к указанному типу данных, если листинг преуспевает; в противном случае возвращает null.

Выражение - значение источника, которое будет передаваться.
Data_type - Тип целевой информации, которую будет выдавать исходное значение.
Длина - это необязательный параметр, указывающий длину целевого типа данных.

```
DECLARE @sampletext AS VARCHAR(10);
SET @sampletext = '123456';

SELECT TRY_CAST(@sampletext AS INT); -- 123456
SELECT TRY_CAST(@sampletext AS DATE); -- NULL
```

В ролях

Функция Cast () используется для преобразования переменной типа данных или данных из одного типа данных в другой тип данных.

Синтаксис

CAST ([выражение] AS Тип данных)

Тип данных, к которому вы выполняете выражение, является типом цели. Тип данных выражения, из которого вы выполняете литье, является типом источника.

```
DECLARE @A varchar(2)
DECLARE @B varchar(2)

set @A='25a'
set @B='15'

Select CAST(@A as int) + CAST(@B as int)  as Result
--'25a' is casted to 25 (string to int)
--'15' is casted to 15 (string to int)

--Result
 --40

DECLARE @C varchar(2)  = 'a'

select CAST(@C as int) as Result
--Result
 --Conversion failed when converting the varchar value 'a' to data type int.
```

Выдает ошибку, если не удалось

Перерабатывать

Когда вы конвертируете выражения из одного типа в другой, во многих случаях в хранимой процедуре или другой подпрограмме потребуется преобразовать данные из типа datetime в тип varchar. Функция Convert используется для таких вещей. Функция CONVERT () может использоваться для отображения данных даты и времени в различных форматах.
Синтаксис

CONVERT (data_type (длина), выражение, стиль)

Значения стиля стиля для преобразования даты или времени в символьные данные. Добавьте 100 к значению стиля, чтобы получить четырехместный год, который включает в себя век (yyyy).

```
select convert(varchar(20),GETDATE(),108)

13:27:16
```

Прочитайте Преобразование типов данных онлайн: https://riptutorial.com/ru/sql-server/topic/5034/преобразование-типов-данных

глава 76: Привилегии или разрешения

Examples

Простые правила

Предоставление разрешения на создание таблиц

```
USE AdventureWorks;
GRANT CREATE TABLE TO MelanieK;
GO
```

Предоставление разрешения SHOWPLAN для роли приложения

```
USE AdventureWorks2012;
GRANT SHOWPLAN TO AuditMonitor;
GO
```

Предоставление CREATE VIEW с опцией GRANT

```
USE AdventureWorks2012;
GRANT CREATE VIEW TO CarmineEs WITH GRANT OPTION;
GO
```

Предоставление всех прав пользователю в конкретной базе данных

```
use YourDatabase
go
exec sp_addrolemember 'db_owner', 'UserName'
go
```

Прочитайте Привилегии или разрешения онлайн: https://riptutorial.com/ru/sql-server/topic/5333/привилегии-или-разрешения

глава 77: применять крест

Examples

Объединение строк таблицы с динамически сгенерированными строками из ячейки

CROSS APPLY позволяет вам «присоединяться» к строкам из таблицы с динамически сгенерированными строками, возвращаемыми некоторой функцией табличного значения.

Представьте, что у вас есть таблица компании со столбцом, содержащим массив продуктов (столбец ProductList), и функцию, которая анализирует эти значения и возвращает набор продуктов. Вы можете выбрать все строки из таблицы компании, применить эту функцию в столбце ProductList и «сгенерировать результаты» с родительской строкой компании:

```
SELECT *
FROM Companies c
     CROSS APPLY dbo.GetProductList( c.ProductList ) p
```

Для каждой строки значение функции *ProductList* будет предоставлено функции, и функция вернет эти продукты в виде набора строк, которые могут быть соединены с родительской строкой.

Присоединить строки таблицы с массивом JSON, хранящимся в ячейке

CROSS APPLY позволяет вам «присоединяться» к строкам из таблицы с коллекцией объектов JSON, хранящихся в столбце.

Представьте, что у вас есть таблица компании со столбцом, содержащим массив продуктов (столбец ProductList), отформатированный как массив JSON. Функция значения функции OPENJSON может анализировать эти значения и возвращать набор продуктов. Вы можете выбрать все строки из таблицы компании, проанализировать продукты JSON с помощью OPENJSON и «сгенерировать результаты» с родительской строкой компании:

```
SELECT *
FROM Companies c
     CROSS APPLY OPENJSON( c.ProductList )
                 WITH ( Id int, Title nvarchar(30), Price money)
```

Для каждой строки значение ячейки *ProductList* будет предоставлено функции OPENJSON, которая преобразует объекты JSON в строки со схемой, определенной в предложении WITH.

Фильтровать строки по значениям массива

Если вы храните список тегов в строке как значения, *разделенные комой,* функция *STRING_SPLIT* позволяет вам преобразовать список тегов в таблицу значений. **CROSS APPLY** позволяет вам «присоединить» значения, проанализированные функцией *STRING_SPLIT,* с родительской строкой.

Представьте, что у вас есть таблица Product с столбцом, который содержит массив тегов, разделенных запятыми (например, promo, sales, new). STRING_SPLIT и CROSS APPLY позволяют вам присоединяться к рядам продуктов с их тегами, чтобы вы могли фильтровать продукты по тегам:

```
SELECT *
FROM Products p
     CROSS APPLY STRING_SPLIT( p.Tags, ',' ) tags
WHERE tags.value = 'promo'
```

Для каждой строки значение ячейки *меток* будет предоставлено функции STRING_SPLIT, которая вернет значения тегов. Затем вы можете фильтровать строки по этим значениям.

Примечание. Функция *STRING_SPLIT* недоступна до **SQL Server 2016**

Прочитайте применять крест онлайн: https://riptutorial.com/ru/sql-server/topic/5462/применять-крест

глава 78: Присоединиться

Вступление

В языке структурированных запросов (SQL) JOIN представляет собой метод связывания двух таблиц данных в одном запросе, позволяя базе данных возвращать набор, содержащий данные из обеих таблиц сразу, или используя данные из одной таблицы, которые будут использоваться как Фильтр по второй таблице. Существует несколько типов JOIN, определенных в стандарте ANSI SQL.

Examples

Внутреннее соединение

`Inner join` возвращает только те записи / строки, которые соответствуют / существуют в обеих таблицах на основе одного или нескольких условий (заданных с помощью ключевого слова `ON`). Это наиболее распространенный тип соединения. Общий синтаксис для `inner join` :

```
SELECT *
FROM table_1
INNER JOIN table_2
  ON table_1.column_name = table_2.column_name
```

Он также может быть упрощен как просто `JOIN` :

```
SELECT *
FROM table_1
JOIN table_2
  ON table_1.column_name = table_2.column_name
```

пример

```
/* Sample data. */
DECLARE @Animal table (
    AnimalId Int IDENTITY,
    Animal Varchar(20)
);

DECLARE @AnimalSound table (
    AnimalSoundId Int IDENTITY,
    AnimalId Int,
    Sound Varchar(20)
);

INSERT INTO @Animal (Animal) VALUES ('Dog');
INSERT INTO @Animal (Animal) VALUES ('Cat');
INSERT INTO @Animal (Animal) VALUES ('Elephant');
```

```
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (1, 'Barks');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (2, 'Meows');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (3, 'Trumpets');
/* Sample data prepared. */

SELECT
    *
FROM
    @Animal
    JOIN @AnimalSound
        ON @Animal.AnimalId = @AnimalSound.AnimalId;
```

```
AnimalId    Animal               AnimalSoundId AnimalId    Sound
----------- -------------------- ------------- ----------- --------------------
1           Dog                  1             1           Barks
2           Cat                  2             2           Meows
3           Elephant             3             3           Trumpets
```

Использование внутреннего соединения с левым внешним соединением (Substitute for Not exists)

Этот запрос вернет данные из таблицы 1, где поля, совпадающие с таблицей 2, с ключом и данными, не указанными в таблице 1, сравниваются с таблицей 2 с условием и ключом

```
select *
  from Table1 t1
    inner join Table2 t2 on t1.ID_Column = t2.ID_Column
    left  join Table3 t3 on t1.ID_Column = t3.ID_Column
  where t2.column_name = column_value
    and t3.ID_Column is null
  order by t1.column_name;
```

Крест

`A cross join` - это декартово объединение, означающее декартово произведение обеих таблиц. Это соединение не требует каких-либо условий для объединения двух таблиц. Каждая строка в левой таблице будет соединяться с каждой строкой правой таблицы. Синтаксис для кросс-соединения:

```
SELECT * FROM table_1
CROSS JOIN table_2
```

Пример:

```
/* Sample data. */
DECLARE @Animal table (
    AnimalId Int IDENTITY,
    Animal Varchar(20)
);

DECLARE @AnimalSound table (
    AnimalSoundId Int IDENTITY,
```

```
    AnimalId Int,
    Sound Varchar(20)
);

INSERT INTO @Animal (Animal) VALUES ('Dog');
INSERT INTO @Animal (Animal) VALUES ('Cat');
INSERT INTO @Animal (Animal) VALUES ('Elephant');

INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (1, 'Barks');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (2, 'Meows');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (3, 'Trumpet');
/* Sample data prepared. */

SELECT
    *
FROM
    @Animal
    CROSS JOIN @AnimalSound;
```

Результаты:

```
AnimalId    Animal              AnimalSoundId AnimalId    Sound
----------- ------------------- ------------- ----------- --------------------
1           Dog                 1             1           Barks
2           Cat                 1             1           Barks
3           Elephant            1             1           Barks
1           Dog                 2             2           Meows
2           Cat                 2             2           Meows
3           Elephant            2             2           Meows
1           Dog                 3             3           Trumpet
2           Cat                 3             3           Trumpet
3           Elephant            3             3           Trumpet
```

Обратите внимание, что есть другие способы, которыми может применяться CROSS JOIN. Это соединение *«старого стиля»* (устарело с ANSI SQL-92) без каких-либо условий, что приводит к перекрестному / декартовому соединению:

```
SELECT *
FROM @Animal, @AnimalSound;
```

Этот синтаксис также работает из-за условия «всегда истинного» соединения, но не рекомендуется и его следует избегать в пользу явного синтаксиса `CROSS JOIN` для удобства чтения.

```
SELECT *
FROM
    @Animal
    JOIN @AnimalSound
        ON 1=1
```

Outer Join

Левая внешняя связь

`LEFT JOIN` возвращает все строки из левой таблицы, соответствующие строкам из правой таблицы, где выполняются условия предложения `ON` . Строки, в которых условие `ON` не выполнено, имеют `NULL` во всех столбцах правой таблицы. Синтаксис `LEFT JOIN` :

```
SELECT * FROM table_1 AS t1
LEFT JOIN table_2 AS t2 ON t1.ID_Column = t2.ID_Column
```

Правостороннее соединение

`RIGHT JOIN` возвращает все строки из правой таблицы, соответствующие строкам из левой таблицы, где выполняются условия предложения `ON` . Строки, в которых условие `ON` не выполнено, имеют `NULL` во всех столбцах левой таблицы. Синтаксис `RIGHT JOIN` :

```
SELECT * FROM table_1 AS t1
RIGHT JOIN table_2 AS t2 ON t1.ID_Column = t2.ID_Column
```

Полная внешняя связь

`FULL JOIN` объединяет `LEFT JOIN` и `RIGHT JOIN` . Все строки возвращаются из обеих таблиц, независимо от того, выполнены ли условия в предложении `ON` . Строки, которые не удовлетворяют условию `ON` , возвращаются с `NULL` во всех столбцах противоположной таблицы (то есть для строки в левой таблице все столбцы в правой таблице будут содержать `NULL` и наоборот). Синтаксис `FULL JOIN` :

```
SELECT * FROM table_1 AS t1
FULL JOIN table_2 AS t2 ON t1.ID_Column = t2.ID_Column
```

Примеры

```
/* Sample test data. */
DECLARE @Animal table (
    AnimalId Int IDENTITY,
    Animal Varchar(20)
);

DECLARE @AnimalSound table (
    AnimalSoundId Int IDENTITY,
    AnimalId Int,
    Sound Varchar(20)
);

INSERT INTO @Animal (Animal) VALUES ('Dog');
INSERT INTO @Animal (Animal) VALUES ('Cat');
INSERT INTO @Animal (Animal) VALUES ('Elephant');
INSERT INTO @Animal (Animal) VALUES ('Frog');

INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (1, 'Barks');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (2, 'Meows');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (3, 'Trumpet');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (5, 'Roars');
/* Sample data prepared. */
```

ЛЕВЫЙ ВНЕШНИЙ ВСТУПИТЕЛЬ

```
SELECT *
FROM @Animal As t1
LEFT JOIN @AnimalSound As t2 ON t1.AnimalId = t2.AnimalId;
```

Результаты для `LEFT JOIN`

```
AnimalId    Animal               AnimalSoundId AnimalId    Sound
----------- -------------------- ------------- ----------- --------------------
1           Dog                  1             1           Barks
2           Cat                  2             2           Meows
3           Elephant             3             3           Trumpet
4           Frog                 NULL          NULL        NULL
```

ПРАВО НА ВСТУПЛЕНИЕ

```
SELECT *
FROM @Animal As t1
RIGHT JOIN @AnimalSound As t2 ON t1.AnimalId = t2.AnimalId;
```

Результаты для `RIGHT JOIN`

```
AnimalId    Animal               AnimalSoundId AnimalId    Sound
----------- -------------------- ------------- ----------- --------------------
1           Dog                  1             1           Barks
2           Cat                  2             2           Meows
3           Elephant             3             3           Trumpet
NULL        NULL                 4             5           Roars
```

ПОЛНОЕ ВНЕШНЕЕ СОЕДИНЕНИЕ

```
SELECT *
FROM @Animal As t1
FULL JOIN @AnimalSound As t2 ON t1.AnimalId = t2.AnimalId;
```

Результаты для `FULL JOIN`

```
AnimalId    Animal               AnimalSoundId AnimalId    Sound
----------- -------------------- ------------- ----------- --------------------
1           Dog                  1             1           Barks
2           Cat                  2             2           Meows
3           Elephant             3             3           Trumpet
4           Frog                 NULL          NULL        NULL
NULL        NULL                 4             5           Roars
```

Использование подключения к обновлению

Соединения также могут использоваться в `UPDATE` :

```
CREATE TABLE Users (
```

```
    UserId int NOT NULL,
    AccountId int NOT NULL,
    RealName nvarchar(200) NOT NULL
)

CREATE TABLE Preferences (
    UserId int NOT NULL,
    SomeSetting bit NOT NULL
)
```

Обновите столбец `SomeSetting` фильтрации таблицы `Preferences` предикатом в таблице `Users` следующим образом:

```
UPDATE p
SET p.SomeSetting = 1
FROM Users u
JOIN Preferences p ON u.UserId = p.UserId
WHERE u.AccountId = 1234
```

`p` является псевдонимом для `Preferences` , определенного в `FROM` пункта заявления. Будут обновляться только строки с соответствующим `AccountId` из таблицы `Users` .

Обновление с помощью левых внешних операторов соединения

```
Update t
SET  t.Column1=100
FROM Table1 t LEFT JOIN Table12 t2
ON t2.ID=t.ID
```

Обновление таблиц с внутренней функцией объединения и агрегата

```
UPDATE t1
SET t1.field1 = t2.field2Sum
FROM table1 t1
INNER JOIN (select field3, sum(field2) as field2Sum
from table2
group by field3) as t2
on t2.field3 = t1.field3
```

Присоединиться к подзапросу

Объединение в подзапрос часто используется, когда вы хотите получить агрегированные данные (например, Count, Avg, Max или Min) из таблицы child / details и отобразить их вместе с записями из таблицы parent / header. Например, вы можете захотеть получить верхнюю / первую дочернюю строку на основе даты или идентификатора, или, возможно, вам нужен граф всех дочерних строк или средний.

В этом примере используются псевдонимы, которые упрощают чтение запросов при использовании нескольких таблиц. В этом случае мы извлекаем все строки из родительской таблицы «Заказы на поставку» и извлекаем только последнюю (или самую

последнюю) дочернюю строку из дочерней таблицы PurchaseOrderLineItems. В этом примере предполагается, что дочерняя таблица использует инкрементные числовые идентификаторы.

```
SELECT po.Id, po.PODate, po.VendorName, po.Status, item.ItemNo,
  item.Description, item.Cost, item.Price
FROM PurchaseOrders po
LEFT JOIN
     (
       SELECT l.PurchaseOrderId, l.ItemNo, l.Description, l.Cost, l.Price, Max(l.id) as Id
       FROM PurchaseOrderLineItems l
       GROUP BY l.PurchaseOrderId, l.ItemNo, l.Description, l.Cost, l.Price
     ) AS item ON item.PurchaseOrderId = po.Id
```

Self Join

Таблицу можно объединить в себя в так называемом самосоединении, совмещая записи в таблице с другими записями в одной таблице. Self-соединения обычно используются в запросах, где определена иерархия в столбцах таблицы.

Рассмотрим образцы данных в таблице « `Employees` :

Я БЫ	название	Boss_ID
1	боб	3
2	Джим	1
3	Сэм	2

`Boss_ID` каждого сотрудника сопоставляется с `ID` другого сотрудника. Чтобы получить список сотрудников с указанием имени своего босса, таблица может быть объединена сама по себе, используя это сопоставление. Обратите внимание, что присоединение к таблице таким образом требует использования псевдонима (`Bosses` в этом случае) во второй ссылке на таблицу, чтобы отличить себя от исходной таблицы.

```
SELECT Employees.Name,
    Bosses.Name AS Boss
FROM Employees
INNER JOIN Employees AS Bosses
    ON Employees.Boss_ID = Bosses.ID
```

Выполнение этого запроса даст следующие результаты:

название	босс
боб	Сэм

название	босс
Джим	боб
Сэм	Джим

Удалить с помощью Join

Соединения также могут использоваться в инструкции `DELETE` . Учитывая схему следующим образом:

```
CREATE TABLE Users (
    UserId int NOT NULL,
    AccountId int NOT NULL,
    RealName nvarchar(200) NOT NULL
)

CREATE TABLE Preferences (
    UserId int NOT NULL,
    SomeSetting bit NOT NULL
)
```

Мы можем удалять строки из таблицы `Preferences` , фильтруя предикат в таблице `Users` следующим образом:

```
DELETE p
FROM Users u
INNER JOIN Preferences p ON u.UserId = p.UserId
WHERE u.AccountId = 1234
```

Здесь `p` является псевдонимом для `Preferences` определенных в предложении `FROM` инструкции, и мы удаляем только строки, имеющие соответствующий `AccountId` из таблицы `Users` .

Случайное превращение внешнего соединения во внутреннее соединение

Внешние соединения возвращают все строки из одной или обеих таблиц плюс совпадающие строки.

```
Table People
PersonID FirstName
       1 Alice
       2 Bob
       3 Eve

Table Scores
PersonID Subject Score
       1 Math    100
       2 Math     54
       2 Science  98
```

Левое соединение таблиц:

```
Select * from People a
left join Scores b
on a.PersonID = b.PersonID
```

Возвращает:

```
PersonID FirstName PersonID Subject Score
       1 Alice            1 Math    100
       2 Bob              2 Math     54
       2 Bob              2 Science  98
       3 Eve           NULL NULL   NULL
```

Если вы хотите вернуть всех людей, с любыми применимыми математическими оценками, распространенная ошибка заключается в том, чтобы написать:

```
Select * from People a
left join Scores b
on a.PersonID = b.PersonID
where Subject = 'Math'
```

Это удалит Еву из ваших результатов, в дополнение к удалению научной оценки Боба, поскольку `Subject` для нее `NULL` .

Правильный синтаксис для удаления записей без Math при сохранении всех лиц в таблице `People` :

```
Select * from People a
left join Scores b
on a.PersonID = b.PersonID
and b.Subject = 'Math'
```

Прочитайте Присоединиться онлайн: https://riptutorial.com/ru/sql-server/topic/1008/присоединиться

глава 79: Просмотры

замечания

Представления представляют собой хранимые запросы, которые могут запрашиваться как обычные таблицы. Представления не являются частью физической модели базы данных. Любые изменения, которые применяются к источнику данных представления, например таблице, также будут отражены в представлении.

Examples

Создать представление

```
CREATE VIEW dbo.PersonsView
AS
SELECT
    name,
    address
FROM persons;
```

Создать или заменить вид

Этот запрос потеряет представление - если он уже существует - и создайте новый.

```
IF OBJECT_ID('dbo.PersonsView', 'V') IS NOT NULL
    DROP VIEW dbo.PersonsView
GO

CREATE VIEW dbo.PersonsView
AS
SELECT
    name,
    address
FROM persons;
```

Создание представления с привязкой схемы

Если создается представление С СХЕМЫ ОБРАБОТКИ, базовую таблицу (таблицы) нельзя отбросить или изменить таким образом, чтобы они нарушили представление. Например, столбец таблицы, на который ссылается в представлении, не может быть удален.

```
CREATE VIEW dbo.PersonsView
WITH SCHEMABINDING
AS
SELECT
    name,
    address
```

```
FROM dbo.PERSONS  -- database schema must be specified when WITH SCHEMABINDING is present
```

Представления *без* привязки схемы могут прерываться, если их базовые таблицы (таблицы) меняются или удаляются. При запросе разбитого представления появляется сообщение об ошибке. sp_refreshview может использоваться для обеспечения того, чтобы существующие представления без привязки схемы не были нарушены.

Прочитайте Просмотры онлайн: https://riptutorial.com/ru/sql-server/topic/5327/просмотры

глава 80: Пространственные данные

Вступление

Существует 2 типа пространственных данных

Геометрическая система координат X / Y для плоской поверхности

География Широта / Долгота системы координат для изогнутой поверхности (земля). Существует несколько проекций криволинейных поверхностей, поэтому каждая пространственная география должна позволять SQL Server знать, какую проекцию использовать. Обычный идентификатор пространственной привязки (SRID) - 4326, который измеряет расстояния в километрах. Это SRID по умолчанию, используемый в большинстве веб-карт

Examples

POINT

Создает единую точку. Это будет геометрия или география в зависимости от используемого класса.

параметр	подробность
Lat или X	Является выражением float, представляющим x-координату создаваемой Точки
Длинные или Y	Является ли float-выражение, представляющее y-координату создаваемой Точки
строка	Хорошо известный текст (WKB) формы геометрии / географии
двоичный	Хорошо известная двоичная (WKB) геометрия / география
SRID	Является выражением int, представляющим идентификатор пространственной привязки (SRID) экземпляра геометрии / географии, который вы хотите вернуть

```
--Explicit constructor
DECLARE @gm1 GEOMETRY = GEOMETRY::Point(10,5,0)

DECLARE @gg1 GEOGRAPHY = GEOGRAPHY::Point(51.511601,-0.096600,4326)

--Implicit constructor (using WKT - Well Known Text)
DECLARE @gm1 GEOMETRY = GEOMETRY::STGeomFromText('POINT(5 10)', 0)
```

```
DECLARE @gg1 GEOGRAPHY= GEOGRAPHY::STGeomFromText('POINT(-0.096600 51.511601)', 4326)

--Implicit constructor (using WKB - Well Known Binary)
DECLARE @gm1 GEOMETRY = GEOMETRY::STGeomFromWKB(0x0101000000000000000000144000000000000002440,
0)

DECLARE @gg1 GEOGRAPHY= GEOGRAPHY::STGeomFromWKB(0x01010000005F29CB10C7BAB8BFEACC3D247CC14940,
4326)
```

Прочитайте Пространственные данные онлайн: https://riptutorial.com/ru/sql-server/topic/6816/пространственные-данные

глава 81: Разметка

Examples

Извлечение граничных значений раздела

```
SELECT        ps.name AS PartitionScheme
            , fg.name AS [FileGroup]
            , prv.*
            , LAG(prv.Value) OVER (PARTITION BY ps.name ORDER BY ps.name, boundary_id) AS
PreviousBoundaryValue

FROM        sys.partition_schemes ps
INNER JOIN    sys.destination_data_spaces dds
            ON dds.partition_scheme_id = ps.data_space_id
INNER JOIN    sys.filegroups fg
            ON dds.data_space_id = fg.data_space_id
INNER JOIN    sys.partition_functions f
            ON f.function_id = ps.function_id
INNER JOIN    sys.partition_range_values prv
            ON f.function_id = prv.function_id
            AND dds.destination_id = prv.boundary_id
```

Переключение разделов

Согласно этой [странице TechNet Microsoft] [1],

> **Данные разделов** позволяют вам быстро и эффективно управлять и получать подмножества данных, сохраняя целостность всего сбора данных.

Когда вы вызываете следующий запрос, данные физически не перемещаются; изменяются только метаданные о местоположении данных.

```
ALTER TABLE [SourceTable] SWITCH TO [TargetTable]
```

Таблицы должны иметь одинаковые столбцы с одинаковыми типами данных и настройками NULL, они должны быть в одной группе файлов, а новая целевая таблица должна быть пустой. См. Ссылку на страницу выше для получения дополнительной информации о переключении разделов.

[1]: https://technet.microsoft.com/en-us/library/ms191160(v=sql.105).aspx Свойство column `IDENTITY` может отличаться.

Получить таблицы разделов, столбцов, схем, функций, итоговые значения и значения min-max с использованием одного запроса

```
SELECT DISTINCT
    object_name(i.object_id) AS [Object Name],
```

```
    c.name AS [Partition Column],
    s.name AS [Partition Scheme],
    pf.name AS [Partition Function],
    prv.tot AS [Partition Count],
    prv.miVal AS [Min Boundry Value],
    prv.maVal AS [Max Boundry Value]
FROM sys.objects o
INNER JOIN sys.indexes i ON i.object_id = o.object_id
INNER JOIN sys.columns c ON c.object_id = o.object_id
INNER JOIN sys.index_columns ic ON ic.object_id = o.object_id
    AND ic.column_id = c.column_id
    AND ic.partition_ordinal = 1
INNER JOIN sys.partition_schemes s ON i.data_space_id = s.data_space_id
INNER JOIN sys.partition_functions pf ON pf.function_id = s.function_id
OUTER APPLY(SELECT
                COUNT(*) tot, MIN(value) miVal, MAX(value) maVal
            FROM sys.partition_range_values prv
            WHERE prv.function_id = pf.function_id) prv
--WHERE object_name(i.object_id) = 'table_name'
ORDER BY OBJECT_NAME(i.object_id)
```

Просто не комментируйте `where` и замените `table_name actual table name` чтобы просмотреть деталь желаемого объекта.

Прочитайте Разметка онлайн: https://riptutorial.com/ru/sql-server/topic/3212/разметка

глава 82: Разрешения базы данных

замечания

Основной синтаксис:

```
{GRANT| REVOKE | DENY} {PERMISSION_NAME} [ON {SECURABLE}] TO {PRINCIPAL};
```

- {GRANT | REVOKE | DENY} - То, что вы пытаетесь сделать
  - Грант: «Дайте это разрешение заявленному руководителю»
  - Отменить: «Удалите это разрешение от заявленного принципала»
  - Запретить: «Удостоверьтесь, что у указанного принципала никогда не было этого разрешения (т. `DENY SELECT` « `DENY SELECT` »означает, что независимо от каких-либо других разрешений `SELECT` не удастся для этого принципала)
- PERMISSION_NAME - операция, на которую вы пытаетесь повлиять. Это будет зависеть от надежности. Например, для хранимой процедуры `GRANT SELECT` не имеет смысла.
- SECURABLE - имя того, на что вы пытаетесь повлиять на разрешения. Это *необязательно* . Вызов `GRANT SELECT TO [aUser];` вполне приемлемо; это означает «для любого защищаемого, для которого имеет значение разрешение `SELECT` , `GRANT` это разрешение».
- ПРИНЦИП - для кого вы пытаетесь повлиять на разрешения. На уровне базы данных это может быть роль (приложение или база данных) или пользователь (например, для входа в систему или нет).

Examples

Изменение разрешений

```
GRANT SELECT ON [dbo].[someTable] TO [aUser];

REVOKE SELECT ON [dbo].[someTable] TO [aUser];
--REVOKE SELECT [dbo].[someTable] FROM [aUser]; is equivalent

DENY SELECT ON [dbo].[someTable] TO [aUser];
```

СОЗДАТЬ ПОЛЬЗОВАТЕЛЯ

```
--implicitly map this user to a login of the same name as the user
CREATE USER [aUser];

--explicitly mapping what login the user should be associated with
CREATE USER [aUser] FOR LOGIN [aUser];
```

СОЗДАТЬ РОЛЬ

```
CREATE ROLE [myRole];
```

Изменение членства в ролях

```
-- SQL 2005+
exec sp_addrolemember @rolename = 'myRole', @membername = 'aUser';
exec sp_droprolemember @rolename = 'myRole', @membername = 'aUser';

-- SQL 2008+
ALTER ROLE [myRole] ADD MEMBER [aUser];
ALTER ROLE [myRole] DROP MEMBER [aUser];
```

Примечание. Ролевые элементы могут быть любыми принципами уровня базы данных. То есть вы можете добавить роль в качестве члена в другую роль. Кроме того, добавление / удаление членов роли является идемпотентным. То есть попытка добавления / удаления приведет к их присутствию / отсутствию (соответственно) в роли независимо от текущего состояния их членства в ролях.

Прочитайте Разрешения базы данных онлайн: https://riptutorial.com/ru/sql-server/topic/6788/разрешения-базы-данных

глава 83: Разрешения и безопасность

Examples

Назначение прав объекта пользователю

В своей продукции хорошая практика для защиты ваших данных и только для того, чтобы операции над ней осуществлялись с помощью хранимых процедур. Это означает, что ваше приложение не может напрямую запускать операции CRUD с вашими данными и потенциально создавать проблемы. Назначение разрешений - трудоемкая, неудобная и обычно обременительная задача. По этой причине его часто проще использовать некоторую (значительную) мощность, содержащуюся в схеме INFORMATION_SCHEMA er, которая содержится в каждой базе данных SQL Server.

Вместо того, чтобы индивидуально назначать разрешения для пользователя по принципу «штучной еды», просто запустите сценарий ниже, скопируйте вывод и запустите его в окне запроса.

```
SELECT 'GRANT EXEC ON core.' + r.ROUTINE_NAME + ' TO ' + <MyDatabaseUsername>
FROM INFORMATION_SCHEMA.ROUTINES r
WHERE r.ROUTINE_CATALOG = '<MyDataBaseName>'
```

Прочитайте Разрешения и безопасность онлайн: https://riptutorial.com/ru/sql-server/topic/7929/разрешения-и-безопасность

глава 84: Расширенные настройки

Examples

Включить и показать дополнительные параметры

```
Exec sp_configure 'show advanced options' ,1
RECONFIGURE
GO
-- Show all configure
sp_configure
```

Включить резервное сжатие по умолчанию

```
Exec sp_configure 'backup compression default',1
GO
RECONFIGURE;
```

Задать коэффициент заполнения по умолчанию

```
sp_configure 'fill factor', 100;
GO
RECONFIGURE;
```

Сервер должен быть перезапущен до того, как изменение вступит в силу.

Установите интервал восстановления системы

```
USE master;
GO
-- Set recovery every 3 min
EXEC sp_configure 'recovery interval', '3';
RECONFIGURE WITH OVERRIDE;
```

Включить разрешение cmd

```
EXEC sp_configure 'xp_cmdshell', 1
GO
RECONFIGURE
```

Установите максимальный размер памяти сервера

```
USE master
EXEC sp_configure 'max server memory (MB)', 64
RECONFIGURE WITH OVERRIDE
```

Задайте количество задач контрольной точки

```
EXEC sp_configure "number of checkpoint tasks", 4
```

Прочитайте Расширенные настройки онлайн: https://riptutorial.com/ru/sql-server/topic/5185/расширенные-настройки

глава 85: Резервное копирование и восстановление базы данных

Синтаксис

- *База данных* BACKUP DATABASE K *backup_device [, ... n]* WITH *with_options [, ... o]*
- *База данных* RESTORE DATABASE FROM *backup_device [, ... n]* WITH *with_options [, ... o]*

параметры

параметр	подробности
база данных	Имя базы данных для резервного копирования или восстановления
backup_device	Устройство для резервного копирования или восстановления базы данных, например {DISK или TAPE}. Может быть разделен запятыми (,)
with_options	Различные параметры, которые можно использовать во время выполнения операции. Как форматирование диска, на котором будет размещаться резервная копия, или восстановление базы данных с заменой.

Examples

Базовое резервное копирование на диск без параметров

Следующая команда резервирует базу данных *«Пользователи»* в файл *«D: \ DB_Backup»* . Лучше не давать расширения.

```
BACKUP DATABASE Users TO DISK = 'D:\DB_Backup'
```

Базовое восстановление с диска без параметров

Следующая команда восстанавливает базу данных *«Пользователи»* из файла *«D: \ DB_Backup»* .

```
RESTORE DATABASE Users FROM DISK = 'D:\DB_Backup'
```

База данных RESTORE с REPLACE

Когда вы пытаетесь восстановить базу данных с другого сервера, вы можете получить следующую ошибку:

> Ошибка 3154: набор резервных копий содержит резервную копию базы данных, отличной от существующей базы данных.

В этом случае вы должны использовать опцию WITH REPLACE, чтобы заменить базу данных базой данных из резервной копии:

```
RESTORE DATABASE WWIDW
FROM DISK = 'C:\Backup\WideWorldImportersDW-Full.bak'
WITH REPLACE
```

Даже в этом случае вы можете получить ошибки, говорящие о том, что файлы не могут быть расположены по определенному пути:

> Msg 3156, уровень 16, состояние 3, строка 1 Файл «WWI_Primary» не может быть восстановлен в «D: \ Data \ WideWorldImportersDW.mdf». Используйте WITH MOVE для определения допустимого местоположения файла.

Эта ошибка происходит, вероятно, из-за того, что ваши файлы не были помещены в тот же путь к папке, который существует на новом сервере. В этом случае вы должны перенести отдельные файлы базы данных в новое место:

```
RESTORE DATABASE WWIDW
FROM DISK = 'C:\Backup\WideWorldImportersDW-Full.bak'
WITH REPLACE,
MOVE 'WWI_Primary' to 'C:\Data\WideWorldImportersDW.mdf',
MOVE 'WWI_UserData' to 'C:\Data\WideWorldImportersDW_UserData.ndf',
MOVE 'WWI_Log' to 'C:\Data\WideWorldImportersDW.ldf',
MOVE 'WWIDW_InMemory_Data_1' to 'C:\Data\WideWorldImportersDW_InMemory_Data_1'
```

С помощью этого оператора вы можете заменить базу данных всеми файлами базы данных, перемещенными в новое место.

Прочитайте Резервное копирование и восстановление базы данных онлайн: https://riptutorial.com/ru/sql-server/topic/5826/резервное-копирование-и-восстановление-базы-данных

глава 86: Сервисный брокер

Examples

1. Основы

Сервисный брокер - это технология, основанная на асинхронной связи между двумя (или более) объектами. Сервисный брокер состоит из: типов сообщений, контрактов, очередей, сервисов, маршрутов и, по крайней мере, конечных точек экземпляра

Подробнее: https://msdn.microsoft.com/en-us/library/bb522893.aspx

2. Включить сервис-брокер в базе данных.

```
ALTER DATABASE [MyDatabase] SET ENABLE_BROKER WITH ROLLBACK IMMEDIATE;
```

3. Создайте базовую конструкцию брокерских услуг в базе данных (связь с одной базой данных)

```
USE [MyDatabase]

CREATE MESSAGE TYPE [//initiator] VALIDATION = WELL_FORMED_XML;
    GO

CREATE CONTRACT [//call/contract]
(
    [//initiator] SENT BY INITIATOR
)
GO

CREATE QUEUE  InitiatorQueue;
GO

CREATE QUEUE  TargetQueue;
GO

CREATE SERVICE InitiatorService
      ON QUEUE InitiatorQueue
(
     [//call/contract]

)

CREATE SERVICE TargetService
ON QUEUE TargetQueue
(
     [//call/contract]

)

GRANT SEND ON SERVICE::[InitiatorService] TO PUBLIC
```

```
GO

GRANT SEND ON SERVICE::[TargetService] TO PUBLIC
GO
```

Нам не нужен маршрут для одной связи с базой данных.

4. Как отправить базовую связь через сервис-брокера

Для этой демонстрации мы будем использовать сервис-брокер, созданный в другой части этой документации. Упомянутая часть называется **3. Создайте базовую конструкцию брокерских услуг в базе данных (связь с одной базой данных)** .

```
USE [MyDatabase]

DECLARE @ch uniqueidentifier = NEWID()
DECLARE @msg XML

BEGIN DIALOG CONVERSATION @ch
    FROM SERVICE [InitiatorService]
    TO SERVICE 'TargetService'
    ON CONTRACT [//call/contract]
    WITH ENCRYPTION = OFF; -- more possible options

        SET @msg = (
                    SELECT 'HelloThere' "elementNum1"
                    FOR XML PATH(''), ROOT('ExampleRoot'), ELEMENTS XSINIL, TYPE
                    );

SEND ON CONVERSATION @ch MESSAGE TYPE [//initiator] (@msg);
END CONVERSATION @ch;
```

После этого разговора будет ваш msg в TargetQueue

5. Как получить разговор с TargetQueue автоматически

Для этой демонстрации мы будем использовать сервис-брокер, созданный в другой части этой документации. Упомянутая часть называется **3. Создайте базовую конструкцию брокерских услуг в базе данных (связь с одной базой данных)** .

Сначала нам нужно создать процедуру, которая может читать и обрабатывать данные из очереди

```
USE [MyDatabase]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[p_RecieveMessageFromTargetQueue]
```

```
    AS
    BEGIN

    declare
    @message_body xml,
    @message_type_name nvarchar(256),
    @conversation_handle uniqueidentifier,
    @messagetypename nvarchar(256);



   WHILE 1=1
   BEGIN

    BEGIN TRANSACTION
        WAITFOR(
        RECEIVE TOP(1)
        @message_body = CAST(message_body as xml),
        @message_type_name = message_type_name,
        @conversation_handle = conversation_handle,
        @messagetypename = message_type_name
        FROM DwhInsertSmsQueue
        ), TIMEOUT 1000;

         IF (@@ROWCOUNT = 0)
            BEGIN
                ROLLBACK TRANSACTION
                BREAK
            END

         IF (@messagetypename = '//initiator')
             BEGIN

                IF OBJECT_ID('MyDatabase..MyExampleTableHelloThere') IS NOT NULL
                    DROP TABLE dbo.MyExampleTableHelloThere

                SELECT @message_body.value('(/ExampleRoot/"elementNum1")[1]', 'VARCHAR(50)')
AS MyExampleMessage
                INTO dbo.MyExampleTableHelloThere

             END



         IF (@messagetypename = 'http://schemas.microsoft.com/SQL/ServiceBroker/EndDialog')
            BEGIN
                END CONVERSATION @conversation_handle;
            END

    COMMIT TRANSACTION
   END


END
```

Второй шаг: позвольте программе TargetQueue автоматически запускать вашу процедуру:

```
USE [MyDatabase]

ALTER QUEUE [dbo].[TargetQueue] WITH STATUS = ON , RETENTION = OFF ,
```

```
ACTIVATION
 (  STATUS = ON , --activation status
    PROCEDURE_NAME = dbo.p_RecieveMessageFromTargetQueue , --procedure name
    MAX_QUEUE_READERS = 1 , --number of readers
    EXECUTE AS SELF  )
```

Прочитайте Сервисный брокер онлайн: https://riptutorial.com/ru/sql-server/topic/7651/сервисный-брокер

глава 87: Системная база данных - TempDb

Examples

Определить использование TempDb

Следующий запрос предоставит информацию об использовании TempDb. Анализируя подсчеты, вы можете определить, какая вещь влияет на TempDb

```
SELECT
 SUM (user_object_reserved_page_count)*8 as usr_obj_kb,
 SUM (internal_object_reserved_page_count)*8 as internal_obj_kb,
 SUM (version_store_reserved_page_count)*8  as version_store_kb,
 SUM (unallocated_extent_page_count)*8 as freespace_kb,
 SUM (mixed_extent_page_count)*8 as mixedextent_kb
FROM sys.dm_db_file_space_usage
```

Attribute	Meaning
Higher number of user objects	More usage of Temp tables , cursors or temp variables
Higher number of internal objects	Query plan is using a lot of database. Ex: sorting, Group by etc.
Higher number of version stores	Long running transaction or high transaction throughput

Сведения о базе данных TempDB

Ниже запрос может быть использован для получения сведений о базе данных TempDB:

```
USE [MASTER]
SELECT * FROM sys.databases WHERE database_id = 2
```

ИЛИ ЖЕ

```
USE [MASTER]
SELECT * FROM sys.master_files WHERE database_id = 2
```

С помощью ниже DMV вы можете проверить, сколько пространства TempDb использует ваш сеанс. Этот запрос весьма полезен при отладке проблем TempDb

```
SELECT * FROM sys.dm_db_session_space_usage WHERE session_id = @@SPID
```

Прочитайте Системная база данных - TempDb онлайн: https://riptutorial.com/ru/sql-server/topic/4427/системная-база-данных---tempdb

глава 88: Снимки базы данных

замечания

Снимки базы данных - это статический вид базы данных SQL Server, доступный только для чтения, который транзакционно согласуется с исходной базой данных с момента создания моментального снимка.

Снимки базы данных всегда находятся на том же экземпляре сервера, что и исходная база данных. По мере обновления базы данных источника обновляется моментальный снимок базы данных.

Снимок отличается от резервной копии, так как процесс создания моментального снимка мгновен, а моментальный снимок занимает пространство только в том случае, если применяются изменения в исходной базе данных. С другой стороны, резервная копия хранит полную копию данных в момент создания резервной копии.

Кроме того, моментальный снимок дает мгновенную копию только для чтения базы данных, в то время как резервное копирование необходимо восстановить на сервере, чтобы быть читаемым (и как только восстановление может быть записано, также)

Снимки базы данных доступны только в выпусках Enterprise и Developer.

Examples

Создание моментального снимка базы данных

Снимки базы данных - это статическое представление базы данных SQL Server, доступное только для чтения (исходная база данных). Он похож на резервную копию, но доступен как любая другая база данных, поэтому клиент может запрашивать базу данных моментальных снимков.

```
CREATE DATABASE MyDatabase_morning -- name of the snapshot
ON (
     NAME=MyDatabase_data, -- logical name of the data file of the source database
     FILENAME='C:\SnapShots\MySnapshot_Data.ss' -- snapshot file;
)
AS SNAPSHOT OF MyDatabase; -- name of source database
```

Вы также можете создать моментальный снимок базы данных с несколькими файлами:

```
CREATE DATABASE MyMultiFileDBSnapshot ON
    (NAME=MyMultiFileDb_ft, FILENAME='C:\SnapShots\MyMultiFileDb_ft.ss'),
    (NAME=MyMultiFileDb_sys, FILENAME='C:\SnapShots\MyMultiFileDb_sys.ss'),
    (NAME=MyMultiFileDb_data, FILENAME='C:\SnapShots\MyMultiFileDb_data.ss'),
```

```
    (NAME=MyMultiFileDb_indx, FILENAME='C:\SnapShots\MyMultiFileDb_indx.ss')
AS SNAPSHOT OF MultiFileDb;
```

Восстановить моментальный снимок базы данных

Если данные в исходной базе данных повреждены или в базу данных записаны неправильные данные, в некоторых случаях возврат базы данных к снимку базы данных, который предшествует повреждению, может быть подходящей альтернативой восстановлению базы данных из резервной копии.

```
RESTORE DATABASE MYDATABASE FROM DATABASE_SNAPSHOT='MyDatabase_morning';
```

> **Предупреждение.** Это приведет к *удалению всех изменений,* внесенных в исходную базу данных с момента снятия моментального снимка!

Удалить снимок

Вы можете удалить существующие снимки базы данных с помощью инструкции DELETE DATABASE:

```
DROP DATABASE Mydatabase_morning
```

В этом заявлении вы должны указать имя моментального снимка базы данных.

Прочитайте Снимки базы данных онлайн: https://riptutorial.com/ru/sql-server/topic/677/снимки-базы-данных

глава 89: Совокупные функции

Вступление

Совокупные функции в SQL Server выполняют вычисления на наборах значений, возвращая одно значение.

Синтаксис

- AVG (*выражение* [ALL | DISTINCT])
- COUNT (*выражение* [ALL | DISTINCT])
- MAX (*выражение* [ALL | DISTINCT])
- MIN (*выражение* [ALL | DISTINCT])
- SUM (*выражение* [ALL | DISTINCT])

Examples

СУММА ()

Возвращает сумму числовых значений в данном столбце.

У нас есть таблица, как показано на рисунке, которая будет использоваться для выполнения различных агрегатных функций. Имя таблицы - это *Marksheet* .

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

```
Select SUM(MarksObtained) From Marksheet
```

Функция `sum` не рассматривает строки с NULL значением в поле, используемом в качестве параметра

В приведенном выше примере, если у нас есть следующая строка:

```
106    Italian    NULL
```

Эта строка не будет учитываться в расчете суммы

AVG ()

Возвращает среднее числовых значений в данном столбце.

У нас есть таблица, как показано на рисунке, которая будет использоваться для выполнения различных агрегатных функций. Имя таблицы - это *Marksheet* .

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

```
Select AVG(MarksObtained) From Marksheet
```

`average` функция не рассматривает строки с NULL значением в поле, которое используется как параметр

В приведенном выше примере, если у нас есть следующая строка:

```
106    Italian    NULL
```

Эта строка не будет учитываться при среднем расчете

МАКСИМУМ()

Возвращает наибольшее значение в данном столбце.

У нас есть таблица, как показано на рисунке, которая будет использоваться для выполнения различных агрегатных функций. Имя таблицы - это *Marksheet* .

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

```
Select MAX(MarksObtained) From Marksheet
```

MIN ()

Возвращает наименьшее значение в данном столбце.

У нас есть таблица, как показано на рисунке, которая будет использоваться для выполнения различных агрегатных функций. Имя таблицы - это *Marksheet* .

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

```
Select MIN(MarksObtained) From Marksheet
```

COUNT ()

Возвращает общее количество значений в данном столбце.

У нас есть таблица, как показано на рисунке, которая будет использоваться для выполнения различных агрегатных функций. Имя таблицы - это *Marksheet* .

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

```
Select COUNT(MarksObtained) From Marksheet
```

Функция `count` не рассматривает строки с NULL значением в поле, используемом в качестве параметра. Обычно параметр count - * (все поля), поэтому, только если все поля строки имеют значение NULL, эта строка не будет считаться

В приведенном выше примере, если у нас есть следующая строка:

```
106    Italian    NULL
```

Эта строка не будет учитываться при подсчете подсчета

НОТА

Функция `COUNT(*)` возвращает количество строк в таблице. Это значение также может быть получено с использованием константного непустого выражения, которое не содержит ссылок на столбцы, таких как `COUNT(1)` .

пример

```
Select COUNT(1) From Marksheet
```

COUNT (Column_Name) с GROUP BY Column_Name

Большую часть времени мы хотели бы получить общее количество появления значения столбца в таблице, например:

TABLE NAME: ОТЧЕТЫ

ReportName	ReportPrice
Тестовое задание	10.00 $
Тестовое задание	10.00 $
Тестовое задание	10.00 $
Тест 2	11.00 $
Тестовое задание	10.00 $
Тест 3	14.00 $
Тест 3	14.00 $
Тест 4	100.00 $

```
SELECT
    ReportName AS REPORT NAME,
    COUNT(ReportName) AS COUNT
FROM
    REPORTS
GROUP BY
    ReportName
```

ИМЯ ДОКЛАДА	COUNT
Тестовое задание	4
Тест 2	1
Тест 3	2
Тест 4	1

Прочитайте Совокупные функции онлайн: https://riptutorial.com/ru/sql-server/topic/5802/совокупные-функции

глава 90: Создание диапазона дат

параметры

параметр	подробности
@С даты	Инклюзивная нижняя граница сгенерированного диапазона дат.
@На свидание	Включая верхнюю границу сгенерированного диапазона дат.

замечания

Большинство экспертов, похоже, рекомендуют создавать таблицу Dates вместо генерации последовательности на лету. См. Http://dba.stackexchange.com/questions/86435/filling-in-date-holes-in-grouped-by-date-sql-data

Examples

Формирование диапазона дат с рекурсивным CTE

Используя рекурсивный CTE, вы можете создать инклюзивный диапазон дат:

```
Declare  @FromDate    Date = '2014-04-21',
         @ToDate      Date = '2014-05-02'

;With DateCte (Date) As
(
    Select  @FromDate Union All
    Select  DateAdd(Day, 1, Date)
    From    DateCte
    Where   Date < @ToDate
)
Select  Date
From    DateCte
Option  (MaxRecursion 0)
```

Значение по умолчанию `MaxRecursion` равно 100. `MaxRecursion` создания более 100 дат с использованием этого метода потребуется сегмент `Option (MaxRecursion N)` запроса, где `N` - желаемый параметр `MaxRecursion` . Установка этого параметра на `0` устранит ограничение `MaxRecursion` .

Создание диапазона дат с таблицей Tally

Другой способ, которым вы можете создать диапазон дат, заключается в использовании таблицы Tally для создания дат между диапазоном:

```
Declare   @FromDate   Date = '2014-04-21',
          @ToDate     Date = '2014-05-02'

;With
   E1(N) As (Select 1 From (Values (1), (1), (1), (1), (1), (1), (1), (1), (1), (1)) DT(N)),
   E2(N) As (Select 1 From E1 A Cross Join E1 B),
   E4(N) As (Select 1 From E2 A Cross Join E2 B),
   E6(N) As (Select 1 From E4 A Cross Join E2 B),
   Tally(N) As
   (
        Select    Row_Number() Over (Order By (Select Null))
        From    E6
   )
Select   DateAdd(Day, N - 1, @FromDate) Date
From     Tally
Where    N <= DateDiff(Day, @FromDate, @ToDate) + 1
```

Прочитайте Создание диапазона дат онлайн: https://riptutorial.com/ru/sql-server/topic/3232/создание-диапазона-дат

глава 91: СОЗДАТЬ ВИД

Examples

СОЗДАТЬ ВИД

```
CREATE VIEW view_EmployeeInfo
AS
    SELECT EmployeeID,
        FirstName,
        LastName,
        HireDate
    FROM Employee
GO
```

Строки из представлений можно выбрать так же, как и таблицы:

```
SELECT FirstName
FROM view_EmployeeInfo
```

Вы также можете создать представление с вычисленным столбцом. Мы можем изменить вид сверху следующим образом, добавив вычисляемый столбец:

```
CREATE VIEW view_EmployeeReport
AS
    SELECT EmployeeID,
        FirstName,
        LastName,
        Coalesce(FirstName,'') + ' ' + Coalesce(LastName,'') as FullName,
        HireDate
    FROM Employee
GO
```

В этом представлении добавляется дополнительный столбец, который будет отображаться при `SELECT` из него строк. Значения в этом дополнительном столбце будут зависеть от полей `FirstName` и `LastName` в таблице `Employee` и будут автоматически обновляться за кадром при обновлении этих полей.

CREATE VIEW С шифрованием

```
CREATE VIEW view_EmployeeInfo
WITH ENCRYPTION
AS
SELECT EmployeeID, FirstName, LastName, HireDate
FROM Employee
GO
```

СОЗДАЙТЕ СМОТРЕТЬ С ВНУТРЕННЕМ СОЕДИНЕНИЕМ

```
CREATE VIEW view_PersonEmployee
AS
    SELECT P.LastName,
        P.FirstName,
        E.JobTitle
    FROM Employee AS E
    INNER JOIN Person AS P
        ON P.BusinessEntityID = E.BusinessEntityID
GO
```

Представления могут использовать объединения для выбора данных из многочисленных источников, таких как таблицы, табличные функции или даже другие виды. В этом примере используются столбцы FirstName и LastName из таблицы Person и столбца JobTitle из таблицы Employee.

Теперь это представление можно использовать для просмотра всех соответствующих строк для Менеджеров в базе данных:

```
SELECT *
FROM view_PersonEmployee
WHERE JobTitle LIKE '%Manager%'
```

CREATE Indexed VIEW

Чтобы создать представление с индексом, представление должно быть создано с использованием `WITH SCHEMABINDING` слов `WITH SCHEMABINDING` :

```
CREATE VIEW view_EmployeeInfo
WITH SCHEMABINDING
AS
    SELECT EmployeeID,
        FirstName,
        LastName,
        HireDate
    FROM [dbo].Employee
GO
```

Теперь могут быть созданы любые кластерные или некластеризованные индексы:

```
CREATE UNIQUE CLUSTERED INDEX IX_view_EmployeeInfo
ON view_EmployeeInfo
(
     EmployeeID ASC
)
```

Есть некоторые ограничения для индексированных просмотров:

- Определение представления может ссылаться на одну или несколько таблиц в одной и той же базе данных.
- После создания уникального кластерного индекса дополнительные дополнительные

некластерные индексы могут быть созданы против представления.

- Вы можете обновлять данные в базовых таблицах, включая вставки, обновления, удаления и даже усечения.
- Вы не можете изменять базовые таблицы и столбцы. Представление создается с помощью опции СХЕМА СОХРАНЕНИЯ.
- Он не может содержать COUNT, MIN, MAX, TOP, внешние соединения или несколько других ключевых слов или элементов.

Для получения дополнительной информации о создании индексированных представлений вы можете прочитать эту статью MSDN

Сгруппированные VIEW

Сгруппированный VIEW основан на запросе с предложением GROUP BY. Поскольку каждая из групп может иметь более одной строки в базе, из которой она была построена, это обязательно VIEW-запросы только для чтения. Такие VIEW обычно имеют одну или несколько совокупных функций, и они используются для целей отчетности. Они также удобны для устранения недостатков SQL. Рассмотрим ВИД, который показывает самую большую продажу в каждом штате. Запрос прост:

https://www.simple-talk.com/sql/t-sql-programming/sql-view-beyond-the-basics/

```
CREATE VIEW BigSales (state_code, sales_amt_total)
AS SELECT state_code, MAX(sales_amt)
     FROM Sales
    GROUP BY state_code;
```

ПРОСМОТРЕТЬ ИЗОБРАЖЕНИЯ

ПРОСМОТРЫ, основанные на операции UNION или UNION ALL, доступны только для чтения, потому что нет единого способа сопоставить изменение только одной строке в одной из базовых таблиц. Оператор UNION удалит повторяющиеся строки из результатов. Операторы UNION и UNION ALL скрывают, из какой таблицы были получены строки. Такие VIEW должны использовать а, потому что столбцы в UNION [ALL] не имеют собственных имен. Теоретически, UNION двух непересекающихся таблиц, ни один из которых не имеет повторяющихся строк сам по себе, должен быть обновляемым.

https://www.simple-talk.com/sql/t-sql-programming/sql-view-beyond-the-basics/

```
CREATE VIEW DepTally2 (emp_nbr, dependent_cnt)
AS (SELECT emp_nbr, COUNT(*)
      FROM Dependents
     GROUP BY emp_nbr)
   UNION
   (SELECT emp_nbr, 0
```

```
 FROM Personnel AS P2
WHERE NOT EXISTS
     (SELECT *
        FROM Dependents AS D2
       WHERE D2.emp_nbr = P2.emp_nbr));
```

Прочитайте СОЗДАТЬ ВИД онлайн: https://riptutorial.com/ru/sql-server/topic/3815/создать-вид

глава 92: СОРТИРОВАТЬ ПО

замечания

Цель предложения ORDER BY - сортировать данные, возвращаемые запросом.

Важно отметить, что **порядок строк, возвращаемых запросом, не определен, если не существует предложение ORDER BY.**

Подробную информацию о предложении ORDER BY см. В документации MSDN: https://msdn.microsoft.com/en-us/library/ms188385.aspx

Examples

Простое предложение ORDER BY

Используя таблицу Employees , ниже приведен пример возврата столбцов Id, FName и LName в порядке возрастания LName:

```
SELECT Id, FName, LName FROM Employees
ORDER BY LName
```

Возвращает:

Я бы	FName	LName
2	Джон	Джонсон
1	Джеймс	кузнец
4	Джонатон	кузнец
3	Майкл	Williams

Чтобы отсортировать в порядке убывания, добавьте ключевое слово DESC после параметра поля, например, один и тот же запрос в порядке убывания LName:

```
SELECT Id, FName, LName FROM Employees
ORDER BY LName DESC
```

ORDER BY несколькими полями

Несколько полей могут быть указаны для предложения `ORDER BY` в порядке ASCending или DESCending.

Например, используя таблицу http://stackoverflow.com/documentation/sql/280/example-databases/1207/item-sales-table#t=20160721131406643421 1 , мы можем вернуть запрос, который сортируется по SaleDate в порядке возрастания, и Количество в порядке убывания.

```
SELECT ItemId, SaleDate, Quantity
FROM [Item Sales]
ORDER BY SaleDate ASC, Quantity DESC
```

Обратите внимание, что ключевое слово `ASC` является необязательным, и результаты сортируются по возрастанию по заданному полю по умолчанию.

ORDER BY со сложной логикой

Если мы хотим упорядочить данные по-разному для каждой группы, мы можем добавить синтаксис `CASE` в `ORDER BY` . В этом примере мы хотим заказать сотрудников из Департамента 1 по фамилии и сотрудникам из отдела 2 по зарплате.

Я бы	FName	LName	Номер телефона	ManagerID	DepartmentID	Оплата труда	Дата приема на работу
1	Джеймс	кузнец	1234567890	НОЛЬ	1	1000	01-01-2002
2	Джон	Джонсон	2468101214	1	1	400	23-03-2005
3	Майкл	Williams	1357911131	1	2	600	12-05-2009
4	Джонатон	кузнец	1212121212	2	1	500	24-07-2016
5	Сэм	саксонский	1372141312	2	2	400	25-03-2015

```
The following query will provide the required results:
SELECT Id, FName, LName, Salary FROM Employees
ORDER BY Case When DepartmentId = 1 then LName else Salary end
```

Пользовательский заказ

Если вы хотите заказать по столбцу, используя что-то другое, кроме алфавитного / числового порядка, вы можете использовать `case` чтобы указать нужный вам заказ.

`order by Group` :

группа	подсчитывать
Не на пенсии	6
В отставке	4
Всего	10

`order by case group when 'Total' then 1 when 'Retired' then 2 else 3 end` возвращает:

группа	подсчитывать
Всего	10
В отставке	4
Не на пенсии	6

Прочитайте СОРТИРОВАТЬ ПО онлайн: https://riptutorial.com/ru/sql-server/topic/4149/сортировать-по

глава 93: Сортировка / упорядочение строк

Examples

ОСНОВЫ

Сначала давайте настроим таблицу примеров.

```
-- Create a table as an example
CREATE TABLE SortOrder
(
    ID INT IDENTITY PRIMARY KEY,
    [Text] VARCHAR(256)
)
GO

-- Insert rows into the table
INSERT INTO SortOrder ([Text])
SELECT ('Lorem ipsum dolor sit amet, consectetur adipiscing elit')
UNION ALL SELECT ('Pellentesque eu dapibus libero')
UNION ALL SELECT ('Vestibulum et consequat est, ut hendrerit ligula')
UNION ALL SELECT ('Suspendisse sodales est congue lorem euismod, vel facilisis libero
pulvinar')
UNION ALL SELECT ('Suspendisse lacus est, aliquam at varius a, fermentum nec mi')
UNION ALL SELECT ('Praesent tincidunt tortor est, nec consequat dolor malesuada quis')
UNION ALL SELECT ('Quisque at tempus arcu')
GO
```

Помните, что при получении данных, если вы не укажете предложение упорядочения строки (ORDER BY), SQL-сервер не гарантирует сортировку (порядок столбцов) **в любое время** . Действительно, в любое время. И нет смысла спорить об этом, он был показан буквально тысячи раз и по всему Интернету.

Нет ORDER BY == no sorting. Конец истории.

```
-- It may seem the rows are sorted by identifiers,
-- but there is really no way of knowing if it will always work.
-- And if you leave it like this in production, Murphy gives you a 100% that it wont.
SELECT * FROM SortOrder
GO
```

Данные двух направлений можно упорядочить по:

- восходящий (перемещение вверх), используя ASC
- нисходящий (перемещение вниз), используя DESC

```
-- Ascending - upwards
SELECT * FROM SortOrder ORDER BY ID ASC
GO
```

```
-- Ascending is default
SELECT * FROM SortOrder ORDER BY ID
GO

-- Descending - downwards
SELECT * FROM SortOrder ORDER BY ID DESC
GO
```

При заказе текстовым столбцом ((n) char или (n) varchar) обратите внимание, что порядок соответствует сортировке. Для получения дополнительной информации о сортировке найдите тему.

Заказ и сортировка данных могут потреблять ресурсы. Именно здесь удобно создавать индексы. Для получения дополнительной информации об индексах найдите тему.

Существует возможность псевдо-рандомизации порядка строк в вашем наборе результатов. Просто заставьте порядок выглядеть недетерминированным.

```
SELECT * FROM SortOrder ORDER BY CHECKSUM(NEWID())
GO
```

Заказ можно запомнить в хранимой процедуре, и так вы должны это сделать, если это последний шаг манипулирования набором строк, прежде чем показывать его конечному пользователю.

```
CREATE PROCEDURE GetSortOrder
AS
    SELECT *
    FROM SortOrder
    ORDER BY ID DESC
GO

EXEC GetSortOrder
GO
```

Существует ограниченная (и хакерская) поддержка для упорядочения в представлениях SQL Server, но рекомендуется не использовать ее.

```
/* This may or may not work, and it depends on the way
   your SQL Server and updates are installed */
CREATE VIEW VwSortOrder1
AS
    SELECT TOP 100 PERCENT *
    FROM SortOrder
    ORDER BY ID DESC
GO

SELECT * FROM VwSortOrder1
GO

-- This will work, but hey... should you really use it?
CREATE VIEW VwSortOrder2
AS
```

```
    SELECT TOP 99999999 *
    FROM SortOrder
    ORDER BY ID DESC
GO

SELECT * FROM VwSortOrder2
GO
```

Для заказа вы можете использовать имена столбцов, псевдонимы или номера столбцов в ORDER BY.

```
SELECT *
FROM SortOrder
ORDER BY [Text]

-- New resultset column aliased as 'Msg', feel free to use it for ordering
SELECT ID, [Text] + ' (' + CAST(ID AS nvarchar(10)) + ')' AS Msg
FROM SortOrder
ORDER BY Msg

-- Can be handy if you know your tables, but really NOT GOOD for production
SELECT *
FROM SortOrder
ORDER BY 2
```

Я советую не использовать цифры в вашем коде, за исключением случаев, когда вы хотите забыть об этом сразу после его выполнения.

Порядок по делам

Если вы хотите сортировать свои данные численно или в алфавитном порядке, вы можете просто использовать `order by [column]` . Если вы хотите сортировать, используя пользовательскую иерархию, используйте оператор case.

```
Group
-----
Total
Young
MiddleAge
Old
Male
Female
```

Используя базовый `order by` :

```
Select * from MyTable
Order by Group
```

возвращает алфавитный вид, что не всегда желательно:

```
Group
-----
```

```
Female
Male
MiddleAge
Old
Total
Young
```

Добавление оператора case, присваивающего возрастающие числовые значения в порядке сортировки ваших данных:

```
Select * from MyTable
Order by case Group
    when 'Total' then 10
    when 'Male' then 20
    when 'Female' then 30
    when 'Young' then 40
    when 'MiddleAge' then 50
    when 'Old' then 60
    end
```

возвращает данные в указанном порядке:

```
Group
-----
Total
Male
Female
Young
MiddleAge
Old
```

Прочитайте Сортировка / упорядочение строк онлайн: https://riptutorial.com/ru/sql-server/topic/5332/сортировка---упорядочение-строк

глава 94: Спусковой крючок

Вступление

Триггер - это особый тип хранимой процедуры, который выполняется автоматически после возникновения события. Существует три типа триггеров: триггеры языка определения данных и триггеры языка управления данными.

Обычно он привязан к столу и срабатывает автоматически. Вы не можете явно вызвать какой-либо триггер.

Examples

Типы и классификации триггеров

В SQL Server существует две категории триггеров: триггеры DDL и триггеры DML.

Триггеры DDL запускаются в ответ на события Data Definition Language (DDL). Эти события в основном соответствуют операторам Transact-SQL, которые начинаются с ключевых слов `CREATE` , `ALTER` и `DROP` .

DML Triggers запускаются в ответ на события языка манипулирования данными (DML). Эти события соответствуют операторам Transact-SQL, которые начинаются с ключевых слов `INSERT` , `UPDATE` и `DELETE` .

Триггеры DML подразделяются на два основных типа:

1. После триггеров (для триггеров)
    - ПОСЛЕ ВСТАВКИ Триггер.
    - ПОСЛЕ ОБНОВЛЕНИЯ Триггер.
    - ПОСЛЕ УДАЛЕНИЯ Триггера.
2. Вместо триггеров
    - ВМЕСТО ВСТАВКИ Триггер.
    - ВМЕСТО ОБНОВЛЕНИЯ Триггер.
    - INSTEAD OF DELETE Триггер.

Триггеры DML

DML Triggers запускаются как ответ на инструкции dml (`insert` , `update` или `delete`). Триггер dml может быть создан для адресации одного или нескольких событий dml для отдельной таблицы или представления. Это означает, что один триггер dml может

обрабатывать вставку, обновление и удаление записей из определенной таблицы или представления, но в них могут обрабатываться только данные, которые изменяются в этой отдельной таблице или представлении.

DML Triggers предоставляет доступ к `inserted` и `deleted` таблицам, в которых содержится информация о данных, которые были / будут затронуты оператором вставки, обновления или удаления, которые запускали триггер.

Обратите внимание, что триггеры DML основаны на операторах, а не на основе строк. Это означает, что если оператор произвел более одной строки, вставленные или удаленные таблицы будут содержать более одной строки.

Примеры:

```
CREATE TRIGGER tblSomething_InsertOrUpdate ON tblSomething
FOR INSERT
AS

    INSERT INTO tblAudit (TableName, RecordId, Action)
    SELECT 'tblSomething', Id, 'Inserted'
    FROM Inserted

GO

CREATE TRIGGER tblSomething_InsertOrUpdate ON tblSomething
FOR UPDATE
AS

    INSERT INTO tblAudit (TableName, RecordId, Action)
    SELECT 'tblSomething', Id, 'Updated'
    FROM Inserted

GO

CREATE TRIGGER tblSomething_InsertOrUpdate ON tblSomething
FOR DELETE
AS

    INSERT INTO tblAudit (TableName, RecordId, Action)
    SELECT 'tblSomething', Id, 'Deleted'
    FROM Deleted

GO
```

Все приведенные выше примеры добавят записи в tblAudit всякий раз, когда запись добавляется, удаляется или обновляется в tblSomething.

Прочитайте Спусковой крючок онлайн: https://riptutorial.com/ru/sql-server/topic/5032/спусковой-крючок

глава 95: Строковые функции

замечания

Список строковых функций (в алфавитном порядке):

- Ascii
- голец
- CHARINDEX
- Concat
- разница
- Формат
- Оставил
- Len
- ниже
- LTrim
- NCHAR
- PATINDEX
- Quotename
- замещать
- копировать
- Задний ход
- Правильно
- RTrim
- Саундэкс
- Космос
- улица
- String_escape

- String_split
- дрянь
- Substring
- Unicode
- верхний

Examples

Оставил

Возвращает подстроку, начинающуюся с самого левого символа строки и до указанной максимальной длины.

Параметры:

1. символьное выражение. `ntext` выражение может быть любого типа данных, который может быть неявно преобразован в `varchar` или `nvarchar` , за исключением `text` или `ntext`
2. максимальная длина. Целое число от 0 `bigint` максимального значения maxint (9,223,372,036,854,775,807).
   Если параметр максимальной длины отрицательный, будет поднята ошибка.

```
SELECT LEFT('This is my string', 4) -- result: 'This'
```

Если максимальная длина больше количества символов в строке, возвращается строка entier.

```
SELECT LEFT('This is my string', 50) -- result: 'This is my string'
```

Правильно

Возвращает вспомогательную строку, которая является самой правой частью строки, с указанной максимальной длиной.

Параметры:

1. символьное выражение. `ntext` выражение может быть любого типа данных, который может быть неявно преобразован в `varchar` или `nvarchar` , за исключением `text` или `ntext`
2. максимальная длина. Целое число от 0 `bigint` максимального значения maxint (9,223,372,036,854,775,807). Если параметр максимальной длины отрицательный, будет поднята ошибка.

```
SELECT RIGHT('This is my string', 6) -- returns 'string'
```

Если максимальная длина больше количества символов в строке, возвращается строка entier.

```
SELECT RIGHT('This is my string', 50) -- returns 'This is my string'
```

Substring

Возвращает подстроку, которая начинается с символа, который указан в указанном начальном индексе и заданной максимальной длине.

Параметры:

1. Выражение символов. `ntext` выражение может быть любого типа данных, который может быть неявно преобразован в `varchar` или `nvarchar` , за исключением `text` или `ntext` .
2. Начать индекс. Число (`int` или `bigint`), которое указывает начальный индекс запрашиваемой подстроки. (**Примечание:** строки в sql-сервере являются индексом base 1, что означает, что первым символом строки является индекс 1). Это число может быть меньше 1. В этом случае, если сумма начального индекса и максимальной длины больше 0, возвращаемой строкой будет строка, начинающаяся с первого символа символьного выражения и с длиной (начальный индекс + максимальная длина - 1). Если оно меньше 0, пустая строка будет возвращена.
3. Максимальная длина. Целое число от 0 `bigint` максимального значения maxint (9,223,372,036,854,775,807). Если параметр максимальной длины отрицательный, будет поднята ошибка.

```
SELECT SUBSTRING('This is my string', 6, 5) -- returns 'is my'
```

Если максимальная длина + начальный индекс больше количества символов в строке, возвращается строка entier.

```
SELECT SUBSTRING('Hello World',1,100) -- returns 'Hello World'
```

Если начальный индекс больше, чем число символов в строке, возвращается пустая строка.

```
SELECT SUBSTRING('Hello World',15,10) -- returns ''
```

ASCII

Возвращает значение int, представляющее код ASCII самого левого символа строки.

```
SELECT ASCII('t') -- Returns 116
SELECT ASCII('T') -- Returns 84
```

```
SELECT ASCII('This') -- Returns 84
```

Если строка является Unicode, а самый левый символ не ASCII, а представлен в текущей сортировке, может быть возвращено значение больше 127:

```
SELECT ASCII(N'ï') -- returns 239 when `SERVERPROPERTY('COLLATION') =
'SQL_Latin1_General_CP1_CI_AS'`
```

Если строка является Unicode, а самый левый символ не может быть представлен в текущей сортировке, возвращается значение int 63 (которое представляет знак вопроса в ASCII):

```
SELECT ASCII(N'□') -- returns 63
SELECT ASCII(nchar(2039)) -- returns 63
```

CHARINDEX

Возвращает начальный индекс первого вхождения строкового выражения внутри другого строкового выражения.

Список параметров:

1. Строка для поиска (до 8000 символов)
2. Строка для поиска (любой допустимый тип и длина символьных данных, включая двоичный)
3. (Необязательно) для начала. Число типов int или большой int. Если пропущено или меньше 1, поиск начинается с начала строки.

Если строка поиска является `varchar(max)` , `nvarchar(max)` или `varbinary(max)` , то `CHARINDEX` функция возвращает `bigint` значение. В противном случае он вернет `int` .

```
SELECT CHARINDEX('is', 'this is my string') -- returns 3
SELECT CHARINDEX('is', 'this is my string', 4) -- returns 6
SELECT CHARINDEX(' is', 'this is my string') -- returns 5
```

голец

Возвращает символ, представленный кодом ASCII.

```
SELECT CHAR(116) -- Returns 't'
SELECT CHAR(84)  -- Returns 'T'
```

Это можно использовать для введения новой строки / строки `CHAR(10)` , возврата каретки `CHAR(13)` и т. Д. См. AsciiTable.com для справки.

Если значение аргумента не находится между 0 и 255, функция CHAR возвращает `NULL` .

Тип возвращаемых данных функции `CHAR` - `char(1)`

Len

Возвращает количество символов строки.
Примечание: функция `LEN` игнорирует конечные пробелы:

```
SELECT LEN('My string'), -- returns 9
       LEN('My string   '), -- returns 9
       LEN('   My string') -- returns 12
```

Если требуется длина, включая конечные пробелы, существует несколько способов достижения этого, хотя у каждого есть свои недостатки. Один из методов заключается в том, чтобы добавить один символ в строку, а затем использовать `LEN` минус один:

```
DECLARE @str varchar(100) = 'My string   '
SELECT LEN(@str + 'x') - 1 -- returns 12
```

Недостатком этого является то, что тип строковой переменной или столбца имеет максимальную длину, добавление дополнительного символа отбрасывается, а результирующая длина все равно не будет считать конечные пробелы. Чтобы решить эту проблему, следующая модифицированная версия решает проблему и дает правильные результаты во всех случаях за счет небольшого количества дополнительного времени выполнения, и из-за этого (правильные результаты, в том числе с суррогатными парами и разумной скоростью выполнения) представляется лучшей методикой для использования:

```
SELECT LEN(CONVERT(NVARCHAR(MAX), @str) + 'x') - 1
```

Другой метод - использовать функцию `DATALENGTH` .

```
DECLARE @str varchar(100) = 'My string   '
SELECT DATALENGTH(@str) -- returns 12
```

Важно отметить, что `DATALENGTH` возвращает длину в байтах строки в памяти. Это будет отличаться для `varchar` vs. `nvarchar` .

```
DECLARE @str nvarchar(100) = 'My string   '
SELECT DATALENGTH(@str) -- returns 24
```

Вы можете отрегулировать это, разделив длину datalength на длину datalength одного символа (который должен быть одного типа). Пример ниже делает это, а также обрабатывает случай, когда целевая строка оказывается пустой, что позволяет избежать деления на ноль.

```
DECLARE @str nvarchar(100) = 'My string   '
SELECT DATALENGTH(@str) / DATALENGTH(LEFT(LEFT(@str, 1) + 'x', 1)) -- returns 12
```

Однако даже это имеет проблемы в SQL Server 2012 и выше. Это приведет к неправильным результатам, когда строка содержит суррогатные пары (некоторые символы могут занимать больше байтов, чем другие символы в одной строке).

Другой способ - использовать `REPLACE` для преобразования пробелов в непространственный символ и взять `LEN` результата. Это дает правильные результаты во всех случаях, но имеет очень низкую скорость выполнения с длинными строками.

Concat

SQL Server 2012

Возвращает строку, которая является результатом объединения двух или более строк. `CONCAT` принимает два или более аргумента.

```
SELECT CONCAT('This', ' is', ' my', ' string') -- returns 'This is my string'
```

Примечание. В отличие от конкатенации строк с использованием оператора конкатенации строк (`+`) при передаче значения null в функцию `concat` он будет неявно преобразовывать его в пустую строку:

```
SELECT CONCAT('This', NULL, ' is', ' my', ' string'), -- returns 'This is my string'
       'This' + NULL + ' is' + ' my' + ' string' -- returns NULL.
```

Также аргументы нестрокового типа будут неявно преобразованы в строку:

```
SELECT CONCAT('This', ' is my ', 3, 'rd string') -- returns 'This is my 3rd string'
```

Нестроковые переменные типа также будут преобразованы в строковый формат, не нужно вручную скрывать или передавать его в строку:

```
DECLARE @Age INT=23;
SELECT CONCAT('Ram is ', @Age,' years old');  -- returns 'Ram is 23 years old'
```

SQL Server 2012

Старые версии не поддерживают функцию `CONCAT` и вместо этого должны использовать оператор конкатенации строк (`+`). Нестроковые типы должны быть переданы или преобразованы в типы строк, чтобы их конкатенация таким образом.

```
SELECT 'This is the number ' + CAST(42 AS VARCHAR(5)) --returns 'This is the number 42'
```

ниже

Возвращает выражение символа (`varchar` или `nvarchar`) после преобразования всех символов верхнего регистра в нижний регистр.

Параметры:

1. Выражение символов. Любое выражение символов или двоичных данных, которые могут быть неявно преобразованы в `varchar` .

```
SELECT LOWER('This IS my STRING') -- Returns 'this is my string'

DECLARE @String nchar(17) = N'This IS my STRING';
SELECT LOWER(@String) -- Returns 'this is my string'
```

верхний

Возвращает выражение символа (`varchar` или `nvarchar`) после преобразования всех строчных символов в верхний регистр.

Параметры:

1. Выражение символов. Любое выражение символов или двоичных данных, которые могут быть неявно преобразованы в `varchar` .

```
SELECT UPPER('This IS my STRING') -- Returns 'THIS IS MY STRING'

DECLARE @String nchar(17) = N'This IS my STRING';
SELECT UPPER(@String) -- Returns 'THIS IS MY STRING'
```

LTrim

Возвращает выражение символа (`varchar` или `nvarchar`) после удаления всех ведущих пробелов, т. `varchar nvarchar` слева от первого символа пробела.

Параметры:

1. символьное выражение. Любое выражение символов или двоичных данных, которые могут быть неявно преобразованы в `varcher` , кроме `text` , `ntext` и `image` .

```
SELECT LTRIM('    This is my string') -- Returns 'This is my string'
```

RTrim

Возвращает выражение символа (`varchar` или `nvarchar`) после удаления всех завершающих пробелов, т. Е. Пробелы с правого конца строки до тех пор, пока не останется первый небелый пробел.

Параметры:

1. символьное выражение. Любое выражение символов или двоичных данных, которые могут быть неявно преобразованы в `varcher` , кроме `text` , `ntext` и `image` .

```
SELECT RTRIM('This is my string     ') -- Returns 'This is my string'
```

Unicode

Возвращает целочисленное значение, представляющее значение Unicode для первого символа входного выражения.

Параметры:

1. Unicode выражение символов. Любое действительное выражение `nchar` или `nvarchar` .

```
SELECT UNICODE(N'Ɛ') -- Returns 400

DECLARE @Unicode nvarchar(11) = N'Ɛ is a char'
SELECT UNICODE(@Unicode) -- Returns 400
```

NCHAR

Возвращает символы Unicode (`nchar(1)` или `nvarchar(2)`), соответствующие целочисленному аргументу, который он получает, как определено стандартом Unicode.

Параметры:

1. целочисленное выражение. Любое целочисленное выражение, которое является положительным числом от 0 до 65535, или если сопоставление базы данных поддерживает флаг дополнительного символа (CS), поддерживаемый диапазон находится в диапазоне от 0 до 1114111. Если целочисленное выражение не попадает внутрь этого диапазона, значение `null` равно вернулся.

```
SELECT NCHAR(257) -- Returns 'ā'
SELECT NCHAR(400) -- Returns 'Ɛ'
```

Задний ход

Возвращает строковое значение в обратном порядке.

Параметры:

1. строковое выражение. Любая строка или двоичные данные, которые могут быть неявно преобразованы в `varchar` .

```
Select REVERSE('Sql Server') -- Returns 'revreS lqS'
```

PATINDEX

Возвращает начальную позицию первого вхождения указанного шаблона в указанном

выражении.

Параметры:

1. шаблон. Выражение символов содержит последовательность, которую нужно найти. Ограничено Максимальная длина 8000 символов. Подстановочные знаки (`%` , `_`) могут использоваться в шаблоне. Если шаблон не начинается с шаблона, он может соответствовать только тому, что находится в начале выражения. Если он не заканчивается подстановочным знаком, он может соответствовать только тому, что находится в конце выражения.
2. выражение. Любой строковый тип данных.

```
SELECT PATINDEX('%ter%', 'interesting') -- Returns 3.

SELECT PATINDEX('%t_r%t%', 'interesting') -- Returns 3.

SELECT PATINDEX('ter%', 'interesting') -- Returns 0, since 'ter' is not at the start.

SELECT PATINDEX('inter%', 'interesting') -- Returns 1.

SELECT PATINDEX('%ing', 'interesting') -- Returns 9.
```

Космос

Возвращает строку (`varchar`) повторяющихся пробелов.

Параметры:

1. целочисленное выражение. Любое целое выражение, до 8000. Если отрицательный, возвращается `null` . если 0, возвращается пустая строка. (Чтобы вернуть строку длиной более 8000 пробелов, используйте Replicate.

```
SELECT SPACE(-1) -- Returns NULL
SELECT SPACE(0)  -- Returns an empty string
SELECT SPACE(3)  -- Returns '   ' (a string containing 3 spaces)
```

копировать

Повторяется строковое значение определенное количество раз.

Параметры:

1. строковое выражение. Строковое выражение может быть символьной строкой или двоичными данными.
2. целочисленное выражение. Любой целочисленный тип, включая `bigint` . Если отрицательный, возвращается `null` . Если 0, возвращается пустая строка.

```
SELECT REPLICATE('a', -1)  -- Returns NULL

SELECT REPLICATE('a', 0)  -- Returns ''

SELECT REPLICATE('a', 5)  -- Returns 'aaaaa'

SELECT REPLICATE('Abc', 3) -- Returns 'AbcAbcAbc'
```

Примечание. Если строковое выражение не относится к типу `varchar(max)` или `nvarchar(max)` , возвращаемое значение не будет превышать 8000 символов. Репликация прекратится до добавления строки, которая приведет к превышению этого значения:

```
SELECT LEN(REPLICATE('a b c d e f g h i j k l', 350)) -- Returns 7981

SELECT LEN(REPLICATE(cast('a b c d e f g h i j k l' as varchar(max)), 350)) -- Returns 8050
```

замещать

Возвращает строку (`varchar` или `nvarchar`), где все вхождения указанной подстроки заменяются другой подстрокой.

Параметры:

1. строковое выражение. Это строка, которую будет искать. Это может быть тип символов или двоичных данных.
2. шаблон. Это подстрока, которая будет заменена. Это может быть тип символов или двоичных данных. Аргумент шаблона не может быть пустой строкой.
3. замена. Это подстрока, которая заменит подстроку шаблона. Это может быть символ или двоичные данные.

```
SELECT REPLACE('This is my string', 'is', 'XX') -- Returns 'ThXX XX my string'.
```

Заметки:

- Если строковое выражение не относится к типу `varchar(max)` или `nvarchar(max)` , функция `replace` усекает возвращаемое значение в 8000 символов.
- Тип возвращаемых данных зависит от типов входных данных - возвращает `nvarchar` если одно из входных значений - `nvarchar` , или `varchar` противном случае.
- Возвращает `NULL` если любой из входных параметров `NULL`

String_Split

SQL Server 2016

Разделяет строковое выражение с помощью разделителя символов. Обратите внимание, что `STRING_SPLIT()` является табличной функцией и поэтому должна использоваться в предложении `FROM` .

Параметры:

1. строка. Любое выражение типа `char` (`char` , `nchar` , `varchar` или `nvarchar`)
2. Seperator. `nchar(1)` выражение любого типа (`char(1)` , `nchar(1)` , `varchar(1)` или `nvarchar(1)`).

Возвращает одну таблицу столбцов, в которой каждая строка содержит фрагмент строки. Имя столбцов - это `value` , а тип данных - `nvarchar` если любой из параметров либо `nchar` либо `nvarchar` , в противном случае `varchar` .

Следующий пример разбивает строку, используя пробел в качестве разделителя:

```
SELECT value FROM STRING_SPLIT('Lorem ipsum dolor sit amet.', ' ');
```

Результат:

```
value
-----
Lorem
ipsum
dolor
sit
amet.
```

Примечания:

> Функция `STRING_SPLIT` доступна только при уровне совместимости **130** . Если уровень совместимости базы данных ниже 130, SQL Server не сможет найти и выполнить функцию `STRING_SPLIT` . Вы можете изменить уровень совместимости базы данных, используя следующую команду:

```
ALTER DATABASE [database_name] SET COMPATIBILITY_LEVEL = 130
```

SQL Server 2016

Старые версии SQL-сервера не имеют встроенной функции разделения строк. Существует множество пользовательских функций, которые обрабатывают проблему разделения строки. Вы можете прочитать статью Аарона Бертрана « Сплит-струны» правильным способом - или следующий лучший способ всестороннего сравнения некоторых из них.

улица

Возвращает символьные данные (`varchar`), преобразованные из числовых данных.

Параметры:

1. float выражение. Примерный числовой тип данных с десятичной точкой.
2. длина. **необязательный.** Общая длина возвращаемого строкового выражения,

включая цифры, десятичную точку и ведущие пробелы (при необходимости). Значение по умолчанию - 10.
3. десятичный. **необязательный.** Число цифр справа от десятичной точки. Если выше 16, результат будет усечен до шестнадцати мест справа от десятичной точки.

```
SELECT STR(1.2) -- Returns '         1'

SELECT STR(1.2, 3) -- Returns '  1'

SELECT STR(1.2, 3, 2) -- Returns '1.2'

SELECT STR(1.2, 5, 2) -- Returns ' 1.20'

SELECT STR(1.2, 5, 5) -- Returns '1.200'

SELECT STR(1, 5, 2) -- Returns ' 1.00'

SELECT STR(1) -- Returns '         1'
```

Quotename

Возвращает строку Unicode, окруженную разделителями, чтобы сделать ее допустимым идентификатором с разделителем SQL Server.

Параметры:

1. символьная строка. Строка данных Unicode, до 128 символов (`sysname`). Если входная строка длиннее 128 символов, функция возвращает `null` .
2. циферблат. **Необязательно** . Один символ для использования в качестве разделителя. Может быть одинарная кавычка (' или ``), левая или правая скобка (`{` , `[` , `(` , `<` или `>` , `)` , `]` , `}`) или двойная кавычка (`"`). Любое другое значение возвращает null Значение по умолчанию - квадратные скобки.

```
SELECT QUOTENAME('what''s my name?')      -- Returns [what's my name?]

SELECT QUOTENAME('what''s my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what''s my name?', ']') -- Returns [what's my name?]

SELECT QUOTENAME('what''s my name?', '''') -- Returns 'what''s my name?'

SELECT QUOTENAME('what''s my name?', '"') -- Returns "what's my name?"

SELECT QUOTENAME('what''s my name?', ')') -- Returns (what's my name?)
SELECT QUOTENAME('what''s my name?', '(') -- Returns (what's my name?)

SELECT QUOTENAME('what''s my name?', '<') -- Returns <what's my name?>
SELECT QUOTENAME('what''s my name?', '>') -- Returns <what's my name?>

SELECT QUOTENAME('what''s my name?', '{') -- Returns {what's my name?}
SELECT QUOTENAME('what''s my name?', '}') -- Returns {what's my name?}

SELECT QUOTENAME('what''s my name?', '`') -- Returns `what's my name?`
```

Саундэкс

Возвращает четырехсимвольный код (`varchar`) для оценки фонетического сходства двух строк.

Параметры:

1. символьное выражение. Алфавитно-цифровое выражение символьных данных.

Функция soundex создает четырехсимвольный код, основанный на том, как будет звучать выражение символов при произнесении. первый символ - это верхний регистр первого символа параметра, остальные 3 символа - это числа, представляющие буквы в выражении (за исключением a, e, i, o, u, h, w и y, которые игнорируются) ,

```
SELECT SOUNDEX ('Smith') -- Returns 'S530'

SELECT SOUNDEX ('Smythe') -- Returns 'S530'
```

разница

Возвращает целочисленное (`int`) значение, которое указывает разницу между значениями soundex двух выражений символов.

Параметры:

1. символьное выражение 1.
2. символьное выражение 2.

Оба параметра представляют собой буквенно-цифровые выражения символьных данных.

Возвращаемое целое число - это количество символов в значениях soundex одинаковых параметров, поэтому 4 означает, что выражения очень похожи, а 0 означает, что они очень разные.

```
SELECT  SOUNDEX('Green'),  -- G650
        SOUNDEX('Greene'),  -- G650
        DIFFERENCE('Green','Greene') -- Returns 4

SELECT  SOUNDEX('Blotchet-Halls'),  -- B432
        SOUNDEX('Greene'),  -- G650
        DIFFERENCE('Blotchet-Halls', 'Greene') -- Returns 0
```

Формат

SQL Server 2012

Возвращает значение `NVARCHAR` форматированное с указанным форматом и культурой (если указано). Это в первую очередь используется для преобразования типов даты в строки.

Параметры:

1. `value` . Выражение поддерживаемого типа данных для форматирования. допустимые типы перечислены ниже.
2. `format` . `NVARCHAR` формата `NVARCHAR` . См. Официальную документацию Microsoft для стандартных и пользовательских форматов.
3. `culture` . **Необязательно** . аргумент `nvarchar` определяющий культуру. Значение по умолчанию - это культура текущего сеанса.

ДАТА

Использование строк стандартного формата:

```
DECLARE @d DATETIME = '2016-07-31';

SELECT
    FORMAT ( @d, 'd', 'en-US' ) AS 'US English Result' -- Returns '7/31/2016'
   ,FORMAT ( @d, 'd', 'en-gb' ) AS 'Great Britain English Result' -- Returns '31/07/2016'
   ,FORMAT ( @d, 'd', 'de-de' ) AS 'German Result' -- Returns '31.07.2016'
   ,FORMAT ( @d, 'd', 'zh-cn' ) AS 'Simplified Chinese (PRC) Result' -- Returns '2016/7/31'
   ,FORMAT ( @d, 'D', 'en-US' ) AS 'US English Result' -- Returns 'Sunday, July 31, 2016'
   ,FORMAT ( @d, 'D', 'en-gb' ) AS 'Great Britain English Result' -- Returns '31 July 2016'
   ,FORMAT ( @d, 'D', 'de-de' ) AS 'German Result' -- Returns 'Sonntag, 31. Juli 2016'
```

Использование строк пользовательского формата:

```
SELECT FORMAT( @d, 'dd/MM/yyyy', 'en-US' ) AS 'DateTime Result' -- Returns '31/07/2016'
      ,FORMAT(123456789,'###-##-####') AS 'Custom Number Result' -- Returns '123-45-6789',
      ,FORMAT( @d,'dddd, MMMM dd, yyyy hh:mm:ss tt','en-US') AS 'US' -- Returns 'Sunday, July
31, 2016 12:00:00 AM'
      ,FORMAT( @d,'dddd, MMMM dd, yyyy hh:mm:ss tt','hi-IN') AS 'Hindi' -- Returns रविवार, जुलाई 31,
2016 12:00:00 पूर्वाह्न
   ,FORMAT ( @d, 'dddd', 'en-US' )  AS 'US' -- Returns 'Sunday'
      ,FORMAT ( @d, 'dddd', 'hi-IN' )  AS 'Hindi' -- Returns 'रविवार'
```

`FORMAT` также может использоваться для форматирования `CURRENCY` , `PERCENTAGE` и `NUMBERS` .

ВАЛЮТА

```
DECLARE @Price1 INT = 40
SELECT FORMAT(@Price1,'c','en-US') AS 'CURRENCY IN US Culture' -- Returns '$40.00'
       ,FORMAT(@Price1,'c','de-DE') AS 'CURRENCY IN GERMAN Culture' -- Returns '40,00 €'
```

Мы можем указать количество цифр после десятичного знака.

```
DECLARE @Price DECIMAL(5,3) = 40.356
SELECT FORMAT( @Price, 'C') AS 'Default', -- Returns '$40.36'
       FORMAT( @Price, 'C0') AS 'With 0 Decimal', -- Returns '$40'
       FORMAT( @Price, 'C1') AS 'With 1 Decimal', -- Returns '$40.4'
       FORMAT( @Price, 'C2') AS 'With 2 Decimal', -- Returns '$40.36'
```

ПРОЦЕНТ

```
DECLARE @Percentage float = 0.35674
SELECT FORMAT( @Percentage, 'P') AS '% Default', -- Returns '35.67 %'
FORMAT( @Percentage, 'P0') AS '% With 0 Decimal', -- Returns '36 %'
FORMAT( @Percentage, 'P1') AS '% with 1 Decimal'  -- Returns '35.7 %'
```

ЧИСЛО

```
DECLARE @Number AS DECIMAL(10,2) = 454545.389
SELECT FORMAT( @Number, 'N','en-US') AS 'Number Format in US', -- Returns '454,545.39'
FORMAT( @Number, 'N','en-IN')  AS 'Number Format in INDIA', -- Returns '4,54,545.39'
FORMAT( @Number, '#.0')     AS 'With 1 Decimal', -- Returns '454545.4'
FORMAT( @Number, '#.00')    AS 'With 2 Decimal', -- Returns '454545.39'
FORMAT( @Number, '#,##.00') AS 'With Comma and 2 Decimal', -- Returns '454,545.39'
FORMAT( @Number, '##.00')   AS 'Without Comma and 2 Decimal', -- Returns '454545.39'
FORMAT( @Number, '000000000') AS 'Left-padded to nine digits' -- Returns '000454545'
```

Список допустимых типов значений: (источник)

```
Category         Type             .Net type
-----------------------------------------
Numeric          bigint           Int64
Numeric          int              Int32
Numeric          smallint         Int16
Numeric          tinyint          Byte
Numeric          decimal          SqlDecimal
Numeric          numeric          SqlDecimal
Numeric          float            Double
Numeric          real             Single
Numeric          smallmoney       Decimal
Numeric          money            Decimal
Date and Time    date             DateTime
Date and Time    time             TimeSpan
Date and Time    datetime         DateTime
Date and Time    smalldatetime    DateTime
Date and Time    datetime2        DateTime
Date and Time    datetimeoffset   DateTimeOffset
```

Важные заметки:

- `FORMAT` возвращает `NULL` для ошибок, отличных от недопустимой культуры. Например, `NULL` возвращается, если значение, указанное в формате, недопустимо.
- `FORMAT` полагается на наличие .NET Framework Common Language Runtime (CLR).
- `FORMAT` опирается на правила форматирования CLR, которые определяют, что двоеточия и периоды должны быть экранированы. Поэтому, когда строка формата (второй параметр) содержит двоеточие или период, двоеточие или период должны быть экранированы с обратным слэшем, когда входное значение (первый параметр) относится к типу данных времени.

См. Также «Форматирование даты и времени» с использованием примера документации FORMAT .

String_escape

SQL Server 2016

Вызывает специальные символы в текстах и возвращает текст (`nvarchar(max)`) с экранированными символами.

Параметры:

1. текст. является выражением `nvarchar` представляющим строку, которая должна быть экранирована.
2. тип. Правила экранирования, которые будут применяться. В настоящее время единственным поддерживаемым значением является `'json'` .

```
SELECT STRING_ESCAPE('\   /
\\    "     ', 'json') -- returns '\\\t\/\n\\\\\t\"\t'
```

Список символов, которые будут экранированы:

```
Special character    Encoded sequence
-------------------------------------
Quotation mark (")   \"
Reverse solidus (\)  \\
Solidus (/)          \/
Backspace            \b
Form feed            \f
New line             \n
Carriage return      \r
Horizontal tab       \t


Control character    Encoded sequence
-------------------------------------
CHAR(0)             \u0000
CHAR(1)             \u0001
...                 ...
CHAR(31)            \u001f
```

Прочитайте Строковые функции онлайн: https://riptutorial.com/ru/sql-server/topic/4113/строковые-функции

глава 96: Студия управления SQL Server (SSMS)

Вступление

SQL Server Management Studio (SSMS) - это инструмент для управления и администрирования SQL Server и базы данных SQL.

Microsoft SSMS бесплатно предоставляется корпорацией Майкрософт.

Имеется документация SSMS .

Examples

Обновление кеша IntelliSense

Когда объекты создаются или изменяются, они не доступны автоматически для IntelliSense. Чтобы сделать их доступными для IntelliSense, необходимо обновить локальный кеш.

В окне редактора запросов нажмите `Ctrl + Shift + R` или выберите « `Edit | IntelliSense | Refresh Local Cache` в меню.

После этого все изменения с момента последнего обновления будут доступны IntelliSense.

Прочитайте Студия управления SQL Server (SSMS) онлайн: https://riptutorial.com/ru/sql-server/topic/10642/студия-управления-sql-server--ssms-

глава 97: Табличные параметры

замечания

Значения параметров таблицы (короткое значение TVP) - это параметры, переданные в хранимую процедуру или функцию, которая содержит данные, которые структурированы в таблице. Использование табличных параметров требует создания определенного пользователем типа таблицы для используемого параметра.

Показанные значения параметров являются параметрами readonly.

Examples

Использование параметра table value для вставки нескольких строк в таблицу

Сначала определите используемый тип таблицы для использования:

```
CREATE TYPE names as TABLE
(
    FirstName varchar(10),
    LastName varchar(10)
)
GO
```

Создайте хранимую процедуру:

```
CREATE PROCEDURE prInsertNames
(
    @Names dbo.Names READONLY -- Note: You must specify the READONLY
)
AS

INSERT INTO dbo.TblNames (FirstName, LastName)
SELECT FirstName, LastName
FROM @Names
GO
```

Выполнение хранимой процедуры:

```
DECLARE @names dbo.Names
INSERT INTO @Names VALUES
('Zohar', 'Peled'),
('First', 'Last')

EXEC dbo.prInsertNames @Names
```

Прочитайте Табличные параметры онлайн: https://riptutorial.com/ru/sql-server/topic/5285/

табличные-параметры

глава 98: Типы данных

Вступление

В этом разделе обсуждаются типы данных, которые может использовать SQL Server, включая их диапазон данных, длину и ограничения (если они есть).

Examples

Точные числа

Существует два основных класса точных числовых типов данных - **Integer** и **Fixed Precision and Scale** .

Целочисленные типы данных

- немного
- TINYINT
- SMALLINT
- ИНТ
- BIGINT

Целые числа - это числовые значения, которые никогда не содержат дробную часть и всегда используют фиксированный объем хранилища. Диапазоны и размеры хранилища целых типов данных показаны в этой таблице:

Тип данных	Спектр	Место хранения
немного	0 или 1	1 бит **
TINYINT	От 0 до 255	1 байт
SMALLINT	-2 ^ 15 (-32,768) до 2 ^ 15-1 (32,767)	2 байта
ИНТ	-2 ^ 31 (-2,147,483,648) до 2 ^ 31-1 (2,147,483,647)	4 байта
BIGINT	-2 ^ 63 (-9,223,372,036,854,775,808) до 2 ^ 63-1 (9,223,372,036,854,775,807)	8 байт

Фиксированные точные и масштабируемые типы данных

- числовой
- десятичный

- smallmoney
- Деньги

Эти типы данных полезны для представления чисел точно. Пока значения могут вписываться в диапазон значений, сохраняемых в типе данных, значение не будет иметь проблемы округления. Это полезно для любых финансовых расчетов, где ошибки округления приведут к клиническому безумию для бухгалтеров.

Обратите внимание, что **десятичные** и **числовые** являются синонимами для одного и того же типа данных.

Тип данных	Спектр	Место хранения
Десятичный [(p [, s])] или числовой [(p [, s])]	-10 ^ 38 + 1 до 10 ^ 38 - 1	См. Таблицу **точности**

При определении *десятичного* или *числового* типа данных вам может потребоваться указать Precision [p] и Scale [s].

Точность - это количество цифр, которое можно сохранить. Например, если вам нужно хранить значения от 1 до 999, вам потребуется точность 3 (чтобы удерживать три цифры в 100). Если вы не указали точность, то по умолчанию используется значение 18.

Масштаб - это количество цифр после десятичной точки. Если вам нужно сохранить число от 0,00 до 999,99, вам нужно указать точность 5 (пять цифр) и шкалу 2 (две цифры после десятичной точки). Вы должны указать точность, чтобы указать масштаб. Шкала по умолчанию равна нулю.

Точность *десятичного* или *числового* типа данных определяет количество байтов, необходимых для хранения значения, как показано ниже:

Точный стол

точность	Байт памяти
1 - 9	5
10-19	9
20-28	13
29-38	17

Денежные фиксированные типы данных

Эти типы данных предназначены специально для учета и других денежных данных. У этих

типов есть фиксированная шкала 4 - вы всегда будете видеть четыре цифры после десятичной точки. Для большинства систем, работающих с большинством валют, будет достаточно *числового* значения с шкалой 2. Обратите внимание, что информация о типе представленной валюты не хранится со значением.

Тип данных	Спектр	Место хранения
Деньги	-922,337,203,685,477,5808 до 922,337,203,685,477.5807	8 байт
smallmoney	-214,748.3648 до 214,748.3647	4 байта

Приблизительная численность

- float [(**n**)]
- реальный

Эти типы данных используются для хранения чисел с плавающей запятой. Поскольку эти типы предназначены для размещения только приблизительных числовых значений, они не должны использоваться в тех случаях, когда любая ошибка округления неприемлема. Однако, если вам нужно обрабатывать очень большие числа или цифры с неопределенным числом цифр после десятичного знака, это может быть вашим лучшим вариантом.

Тип данных	Спектр	Размер
поплавок	-1,79E + 308-2,23E-308, 0 и 2,23E-308 до 1,79E + 308	зависит от **n** в таблице ниже
реальный	-3.40E + 38 до -1.18E - 38, 0 и 1.18E - от 38 до 3.40E + 38	4 байта

n таблица значений для чисел с *плавающей точкой* . Если в объявлении float не указано значение, будет использоваться значение по умолчанию 53. Обратите внимание, что *float (24)* является эквивалентом *реального* значения.

n значение	точность	Размер
1-24	7 цифр	4 байта
25-53	15 цифр	8 байт

Дата и время

Эти типы находятся во всех версиях SQL Server

- Дата и время
- smalldatetime

Эти типы находятся во всех версиях SQL Server после SQL Server 2012

- Дата
- DateTimeOffset
- datetime2
- время

Строки символов

- голец
- VARCHAR
- текст

Строки символов Unicode

- NCHAR
- NVARCHAR
- NTEXT

Бинарные строки

- двоичный
- VARBINARY
- образ

Другие типы данных

- курсор
- отметка времени
- `hierarchyid`
- уникальный идентификатор
- sql_variant
- XML
- Таблица
- Пространственные типы

Прочитайте Типы данных онлайн: https://riptutorial.com/ru/sql-server/topic/5260/типы-данных

глава 99: Типы пользовательских таблиц

Вступление

Определенные пользователем типы таблиц (для краткости - UDT) - это типы данных, которые позволяют пользователю определять структуру таблицы. Определенные пользователем типы таблиц поддерживают первичные ключи, уникальные ограничения и значения по умолчанию.

замечания

UDT имеют следующие ограничения:

- не может использоваться как столбец в таблице или поле в структурированных пользовательских типах
- некластеризованный индекс не может быть создан в UDT, если индекс не является результатом создания ограничения PRIMARY KEY или UNIQUE на UDT
- Определение UDT НЕ МОЖЕТ быть изменено после его создания

Examples

создание UDT с одним столбцом int, который также является первичным ключом

```
CREATE TYPE dbo.Ids as TABLE
(
    Id int PRIMARY KEY
)
```

Создание UDT с несколькими столбцами

```
CREATE TYPE MyComplexType as TABLE
(
    Id int,
    Name varchar(10)
)
```

Создание UDT с уникальным ограничением:

```
CREATE TYPE MyUniqueNamesType as TABLE
(
    FirstName varchar(10),
    LastName varchar(10),
    UNIQUE (FirstName,LastName)
```

```
)
```

Примечание. Ограничения в пользовательских типах таблиц нельзя назвать.

Создание UDT с первичным ключом и столбцом со значением по умолчанию:

```
CREATE TYPE MyUniqueNamesType as TABLE
(
    FirstName varchar(10),
    LastName varchar(10),
    CreateDate datetime default GETDATE()
    PRIMARY KEY (FirstName,LastName)
)
```

Прочитайте Типы пользовательских таблиц онлайн: https://riptutorial.com/ru/sql-server/topic/5280/типы-пользовательских-таблиц

глава 100: Управление базами данных Azure SQL

Examples

Найти информацию уровня сервиса для базы данных Azure SQL

База данных Azure SQL имеет разные версии и уровни производительности.

Вы можете найти версию, версию (базовую, стандартную или премиальную) и цель обслуживания (S0, S1, P4, P11 и т. Д.) Базы данных SQL, которая работает как служба в Azure, используя следующие инструкции:

```
select @@version
SELECT DATABASEPROPERTYEX('Wwi', 'EDITION')
SELECT DATABASEPROPERTYEX('Wwi', 'ServiceObjective')
```

Изменить уровень обслуживания базы данных Azure SQL

Вы можете масштабировать или масштабировать базу данных Azure SQL с помощью инструкции ALTER DATABASE:

```
ALTER DATABASE WWI
MODIFY (SERVICE_OBJECTIVE = 'P6')
-- or
ALTER DATABASE CURRENT
MODIFY (SERVICE_OBJECTIVE = 'P2')
```

Если вы попытаетесь изменить уровень обслуживания при изменении уровня обслуживания текущей базы данных, все еще выполняется, вы получите следующую ошибку:

> Msg 40802, уровень 16, состояние 1, строка 1 Назначение служебных целей на сервере «.....» и база данных «.....» уже выполняется. Подождите, пока состояние назначения цели службы для базы данных будет помечено как «Завершено».

Повторно запустите инструкцию ALTER DATABASE, когда заканчивается переходный период.

Репликация базы данных Azure SQL

Вы можете создать вторичную копию базы данных с тем же именем на другом сервере Azure SQL Server, сделав локальную базу данных первичной и начните асинхронную

репликацию данных с первичной на новую вторичную.

```
ALTER DATABASE <<mydb>>
ADD SECONDARY ON SERVER <<secondaryserver>>
WITH ( ALLOW_CONNECTIONS = ALL )
```

Целевой сервер может находиться в другом центре обработки данных (используется для георепликации). Если база данных с тем же именем уже существует на целевом сервере, команда завершится с ошибкой. Команда выполняется в основной базе данных на сервере, на котором размещена локальная база данных, которая станет основной. Если для ALLOW_CONNECTIONS установлено значение ALL (по умолчанию установлено значение NO), вторичная реплика будет представлять собой базу данных только для чтения, которая позволит подключиться ко всем входам с соответствующими разрешениями.

Вторичную репликацию базы данных можно повысить до первичной, используя следующую команду:

```
ALTER DATABASE mydb FAILOVER
```

Вы можете удалить вторичную базу данных на вторичном сервере:

```
ALTER DATABASE <<mydb>>
REMOVE SECONDARY ON SERVER <<testsecondaryserver>>
```

Создание базы данных Azure SQL в эластичном пуле

Вы можете поместить свою базу данных SQL в базу данных SQL:

```
CREATE DATABASE wwi
( SERVICE_OBJECTIVE = ELASTIC_POOL ( name = mypool1 ) )
```

Вы можете создать копию существующей базы данных и поместить ее в некоторый эластичный пул:

```
CREATE DATABASE wwi
AS COPY OF myserver.WideWorldImporters
( SERVICE_OBJECTIVE = ELASTIC_POOL ( name = mypool1 ) )
```

Прочитайте Управление базами данных Azure SQL онлайн: https://riptutorial.com/ru/sql-server/topic/7113/управление-базами-данных-azure-sql

глава 101: Уровни изоляции и блокировка

замечания

Я нашел эту ссылку - это полезно в качестве ссылки: «Уровни изоляции»

Examples

Примеры установки уровня изоляции

Пример установки уровня изоляции:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SELECT * FROM Products WHERE ProductId=1;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; --return to the default one
```

1. `READ UNCOMMITTED` - означает, что запрос в текущей транзакции не может получить доступ к измененным данным из другой транзакции, которая еще не совершена - никакие грязные чтения! НО, возможны неповторимые чтения и фантомные чтения, поскольку данные могут быть изменены другими транзакциями.

2. `REPEATABLE READ` - означает, что запрос в текущей транзакции не может получить доступ к измененным данным другой транзакции, которая еще не совершена - не грязные чтения! Никакие другие транзакции не могут изменять данные, считываемые текущей транзакцией, до тех пор, пока они не будут завершены, что исключает чтение NONREPEATABLE. НО, если другая транзакция вставляет NEW ROWS и запрос выполняется более одного раза, могут появиться фантомные строки, начиная с второго чтения (если оно соответствует запросу where запроса).

3. `SNAPSHOT` - возможность возврата данных, которые существуют в начале запроса. Обеспечивает согласованность данных. Он предотвращает грязные чтения, неповторяемые чтения и фантомные чтения. Чтобы использовать это - необходима конфигурация DB:

```
ALTER DATABASE DBTestName SET ALLOW_SNAPSHOT_ISOLATION ON;GO;
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
```

4. `READ COMMITTED` - изоляция по умолчанию SQL-сервера. Он предотвращает чтение данных, которые были изменены другой транзакцией, пока не будет совершен. Он использует общую блокировку и управление версиями строк в таблицах, что предотвращает грязные чтения. Это зависит от конфигурации БД READ_COMMITTED_SNAPSHOT - если включено - используется управление версиями строк. для включения - используйте это:

```
ALTER DATABASE DBTestName SET ALLOW_SNAPSHOT_ISOLATION ON;GO;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED; --return to the default one
```

5. `SERIALIZABLE` - использует физические блокировки, которые были приобретены и удерживаются до конца транзакции, что предотвращает грязные чтения, фантомные чтения, неповторяемые чтения. НО, это влияет на производительность базы данных, поскольку параллельные транзакции сериализуются и выполняются один за другим.

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ;
```

Прочитайте Уровни изоляции и блокировка онлайн: https://riptutorial.com/ru/sql-server/topic/5331/уровни-изоляции-и-блокировка

глава 102: Уровни изоляции транзакций

Синтаксис

- УСТАНОВИТЬ УРОВЕНЬ ИЗОЛЯЦИИ СТАВКИ {ПРОЧИТАТЬ НЕОБХОДИМЫЕ | ПРОЧИТАЙТЕ ОБЯЗАТЕЛЬНО | ПОВТОРНЫЙ ПРОЧИТАЙТЕ | SNAPSHOT | SERIALIZABLE} [;]

замечания

Ссылка MSDN: УРОВЕНЬ ИЗОЛЯЦИИ УСТАНОВКИ СТАВКИ

Examples

Читать

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
```

Это самый разрешительный уровень изоляции, поскольку он вообще не вызывает блокировок. Он указывает, что операторы могут читать все строки, включая строки, которые были записаны в транзакциях, но еще не выполнены (т. Е. Они все еще находятся в транзакции). Этот уровень изоляции может подвергаться «грязному чтению».

Чтение переведено

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
```

Этот уровень изоляции является вторым наиболее разрешительным. Он предотвращает грязные чтения. Поведение `READ COMMITTED` зависит от настройки `READ_COMMITTED_SNAPSHOT` :

- Если установлено значение «ВЫКЛ» (значение по умолчанию), транзакция использует совместные блокировки для предотвращения изменения других транзакций в строках, используемых текущей транзакцией, а также блокирует текущую транзакцию от чтения строк, измененных другими транзакциями.
- Если установлено значение ON, `READCOMMITTEDLOCK` таблицы `READCOMMITTEDLOCK` может использоваться для запроса общей блокировки вместо управления версиями строк для транзакций, выполняемых в режиме `READ COMMITTED` .

Примечание: `READ COMMITTED` - это поведение SQL Server по умолчанию.

Что такое «грязное чтение»?

Грязные чтения (или незафиксированные чтения) - это чтения строк, которые изменяются открытой транзакцией.

Это поведение можно реплицировать, используя два отдельных запроса: один для открытия транзакции и записи некоторых данных в таблицу без фиксации, а другой - для выбора данных, которые будут записаны (но еще не зафиксированы) с этим уровнем изоляции.

Запрос 1 - Подготовьте транзакцию, но не завершите ее:

```
CREATE TABLE dbo.demo (
    col1 INT,
    col2 VARCHAR(255)
);
GO
--This row will get committed normally:
BEGIN TRANSACTION;
    INSERT INTO dbo.demo(col1, col2)
    VALUES (99, 'Normal transaction');
COMMIT TRANSACTION;
--This row will be "stuck" in an open transaction, causing a dirty read
BEGIN TRANSACTION;
    INSERT INTO dbo.demo(col1, col2)
    VALUES (42, 'Dirty read');
--Do not COMMIT TRANSACTION or ROLLBACK TRANSACTION here
```

Запрос 2 - Чтение строк, включая открытую транзакцию:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SELECT * FROM dbo.demo;
```

Возвращает:

```
col1        col2
----------- ---------------------------------------
99          Normal transaction
42          Dirty read
```

PS: Не забудьте очистить эти демо-данные:

```
COMMIT TRANSACTION;
DROP TABLE dbo.demo;
GO
```

Повторяемое чтение

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
```

Этот уровень изоляции транзакций несколько менее разрешительный, чем `READ COMMITTED` , поскольку общие блокировки размещаются во всех данных, считанных каждым оператором транзакции, и удерживаются **до завершения транзакции** , а не освобождаются после каждого утверждения.

Примечание. Используйте этот параметр только в случае необходимости, так как это скорее приведет к ухудшению производительности базы данных, а также к блокировкам, чем `READ COMMITTED` .

снимок

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT
```

Указывает, что данные, считываемые любым оператором в транзакции, будут транзакционной последовательностью версий данных, существовавших в начале транзакции, то есть будут считываться только данные, которые были сделаны до начала транзакции.

Операции `SNAPSHOT` не запрашивают или не вызывают блокировки данных, которые читаются, поскольку они только читают версию (или моментальный снимок) данных, которые существовали на момент начала транзакции.

Транзакция, выполняющаяся на уровне изоляции `SNAPSHOT` считывает только свои собственные изменения данных во время ее работы. Например, транзакция может обновлять некоторые строки, а затем читать обновленные строки, но это изменение будет видимым только для текущей транзакции до ее фиксации.

Примечание. Параметр `ALLOW_SNAPSHOT_ISOLATION` должен быть установлен в положение ON до того, как можно использовать уровень изоляции `SNAPSHOT` .

Сериализуемый

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL SERIALIZEABLE
```

Этот уровень изоляции является наиболее ограничительным. Он запрашивает **диапазон, который блокирует** диапазон значений ключей, которые считываются каждым оператором в транзакции. Это также означает, что `INSERT` из других транзакций будут заблокированы, если строки, которые будут вставлены, находятся в диапазоне, заблокированном текущей транзакцией.

Эта опция имеет тот же эффект, что и установка `HOLDLOCK` во всех таблицах во всех `SELECT` транзакции.

Примечание. Эта изоляция транзакций имеет самый низкий уровень параллелизма и должна использоваться только при необходимости.

Прочитайте Уровни изоляции транзакций онлайн: https://riptutorial.com/ru/sql-server/topic/5114/уровни-изоляции-транзакций

глава 103: Установка SQL Server в Windows

Examples

Вступление

Это доступные выпуски SQL Server, как сказано в Матрице выпусков:

- Экспресс: бесплатная база данных начального уровня. Включает функциональность RDBMS ядра. Ограничено до 10G от размера диска. Идеально подходит для разработки и тестирования.
- Стандартная версия: стандартная лицензионная версия. Включает основные функциональные возможности и возможности Business Intelligence.
- Enterprise Edition: полнофункциональная версия SQL Server. Включает расширенные функции безопасности и хранилища данных.
- Developer Edition: включает все функции Enterprise Edition и никаких ограничений, и их можно загружать и использовать только для целей разработки.

После загрузки / приобретения SQL Server установка выполняется с помощью SQLSetup.exe, которая доступна в виде графического интерфейса или программы командной строки.

Для установки через любой из них потребуется указать ключ продукта и запустить некоторую начальную конфигурацию, включающую включенные функции, отдельные службы и настройку начальных параметров для каждого из них. Дополнительные сервисы и функции можно активировать в любое время, запустив программу SQLSetup.exe либо в командной строке, либо в графическом интерфейсе.

Прочитайте Установка SQL Server в Windows онлайн: https://riptutorial.com/ru/sql-server/topic/5801/установка-sql-server-в-windows

глава 104: Функции агрегации строк в SQL Server

Examples

Использование STUFF для объединения строк

У нас есть таблица Student с SubjectId. Здесь требование заключается в конкатенации на основе subjectId.

Все версии SQL Server

```
create table #yourstudent (subjectid int, studentname varchar(10))

insert into #yourstudent (subjectid, studentname) values
 ( 1       ,'Mary'    )
,( 1       ,'John'    )
,( 1       ,'Sam'    )
,( 2       ,'Alaina')
,( 2       ,'Edward')

select subjectid,  stuff(( select concat( ',', studentname) from #yourstudent y where
y.subjectid = u.subjectid for xml path('')),1,1, '')
    from #yourstudent u
    group by subjectid
```

String_Agg для агрегирования строк

В случае SQL Server 2017 или vnext мы можем использовать встроенную STRING_AGG для этой агрегации. Для того же студенческого стола,

```
create table #yourstudent (subjectid int, studentname varchar(10))

insert into #yourstudent (subjectid, studentname) values
 ( 1       ,'Mary'    )
,( 1       ,'John'    )
,( 1       ,'Sam'    )
,( 2       ,'Alaina')
,( 2       ,'Edward')

select subjectid, string_agg(studentname, ',') from #yourstudent
    group by subjectid
```

Прочитайте Функции агрегации строк в SQL Server онлайн: https://riptutorial.com/ru/sql-server/topic/9892/функции-агрегации-строк-в-sql-server

глава 105: Функции окна

Examples

Центрированное скользящее среднее

Рассчитайте скользящее среднее значение по 6-месячному (126-дневному) среднему по цене:

```
SELECT TradeDate, AVG(Px) OVER (ORDER BY TradeDate ROWS BETWEEN 63 PRECEDING AND 63 FOLLOWING)
AS PxMovingAverage
FROM HistoricalPrices
```

Обратите внимание: поскольку в начале и в конце диапазона TradeDate он занимает *до* 63 строк до и после каждой возвращаемой строки, он не будет центрирован: когда он достигнет наибольшего TradeDate, он сможет найти только 63 предшествующих значения включить в среднем.

Найти самый последний элемент в списке событий, отмеченных по времени.

В таблицах, регистрирующих события, часто бывает поле datetime, записывающее время, когда произошло событие. Поиск единственного последнего события может быть затруднен, потому что всегда возможно, что два события были записаны с точно идентичными отметками времени. Вы можете использовать row_number () over (order by ...), чтобы убедиться, что все записи уникально ранжированы и выберите верхнюю (где my_ranking = 1)

```
select *
from (
    select
        *,
        row_number() over (order by crdate desc) as my_ranking
    from sys.sysobjects
) g
where my_ranking=1
```

Этот же метод можно использовать для возврата одной строки из любого набора данных с потенциально повторяющимися значениями.

Скользящее среднее за последние 30 пунктов

Скользящая средняя за последние 30 единиц

```
SELECT
```

```
    value_column1,
    (   SELECT
            AVG(value_column1) AS moving_average
        FROM Table1 T2
        WHERE ( SELECT
                    COUNT(*)
                FROM Table1 T3
                WHERE date_column1 BETWEEN T2.date_column1 AND T1.date_column1
                ) BETWEEN 1 AND 30
    ) as MovingAvg
FROM Table1 T1
```

Прочитайте Функции окна онлайн: https://riptutorial.com/ru/sql-server/topic/3209/функции-окна

глава 106: Функции ранжирования

Синтаксис

- DENSE_RANK () OVER ([<partition_by_clause>] <order_by_clause>)
- RANK () OVER ([partition_by_clause] order_by_clause)

параметры

аргументы	подробности
`<partition_by_clause>`	Разделение набора результатов, созданного предложением FROM , на разделы, к которым `DENSE_RANK` функция `DENSE_RANK` . Для синтаксиса `PARTITION BY` см. Раздел OVER (Transact-SQL) .
`<order_by_clause>`	Определяет порядок, в котором функция `DENSE_RANK` применяется к строкам в разделе.
`OVER ( [ partition_by_clause ] order_by_clause)`	`partition_by_clause` делит результирующий набор, созданный предложением `FROM` , на разделы, к которым применяется функция. Если не указано, функция обрабатывает все строки набора результатов запроса как одну группу. order_by_clause определяет порядок данных перед применением функции. Требуется order_by_clause. Предложение `<rows or range clause>` предложения `OVER` не может быть указано для функции `RANK` . Для получения дополнительной информации см. Раздел OVER (Transact-SQL) .

замечания

Если две или более строки привязаны к рангу в одном разделе, каждая связанная строка получает тот же ранг. Например, если у двух топ-менеджеров есть одинаковое значение SalesYTD, они оба ранжируются. Продавца со следующим самым высоким SalesYTD занимает второе место. Это больше, чем количество отдельных строк, которые появляются перед этой строкой. Поэтому числа, возвращаемые функцией `DENSE_RANK` , не имеют пробелов и всегда имеют последовательные ряды.

Порядок сортировки, используемый для всего запроса, определяет порядок, в котором строки появляются в результате. Это означает, что строка, занявшая первое место, не должна быть первой строкой в разделе.

`DENSE_RANK` является недетерминированным. Для получения дополнительной информации см. Детерминированные и недетерминированные функции .

Examples

РАНГ()

A RANK () Возвращает ранг каждой строки в результирующем наборе секционированного столбца.

Например:

```
Select Studentid,Name,Subject,Marks,
RANK() over(partition by name order by Marks desc)Rank
From Exam
order by name,subject

   Studentid    Name    Subject    Marks    Rank

    101         Ivan    Maths       70       2
    101         Ivan    Science     80       1
    101         Ivan    Social      60       3
    102         Ryan    Maths       60       2
    102         Ryan    Science     50       3
    102         Ryan    Social      70       1
    103         Tanvi   Maths       90       1
    103         Tanvi   Science     90       1
    103         Tanvi   Social      80       3
```

DENSE_RANK ()

То же, что и RANK (). Он возвращает ранг без пробелов:

```
Select  Studentid, Name,Subject,Marks,
DENSE_RANK() over(partition by name order by Marks desc)Rank
From Exam
order by name

Studentid    Name    Subject    Marks    Rank
101          Ivan    Science    80       1
101          Ivan    Maths      70       2
101          Ivan    Social     60       3
102          Ryan    Social     70       1
102          Ryan    Maths      60       2
102          Ryan    Science    50       3
103          Tanvi   Maths      90       1
103          Tanvi   Science    90       1
103          Tanvi   Social     80       2
```

Прочитайте Функции ранжирования онлайн: https://riptutorial.com/ru/sql-server/topic/5031/функции-ранжирования

глава 107: Функция Split String на сервере Sql

Examples

Разделить строку на Sql Server 2016

В **SQL Server 2016,** наконец, они представили функцию строки Split: `STRING_SPLIT`

Параметры: он принимает два параметра

Строка :

> Является выражением любого символьного типа (т.е. nvarchar, varchar, nchar или char).

разделитель :

> Является единственным символьным выражением любого символьного типа (например, nvarchar (1), varchar (1), nchar (1) или char (1)), который используется как разделитель для конкатенированных строк.

Примечание. Вы всегда должны проверить, является ли выражение непустой строкой.

Пример:

```
Select Value
From STRING_SPLIT('a|b|c','|')
```

В приведенном выше примере

```
String    : 'a|b|c'
separator : '|'
```

Результат:

```
+-----+
|Value|
+-----+
|a    |
+-----+
|b    |
+-----+
|c    |
+-----+
```

Если это пустая строка:

```
SELECT value
FROM STRING_SPLIT('',',')
```

Результат:

```
  +-----+
  |Value|
  +-----+
1 |     |
  +-----+
```

Вы можете избежать вышеуказанной ситуации, добавив предложение `WHERE`

```
SELECT value
FROM STRING_SPLIT('',',')
WHERE LTRIM(RTRIM(value))<>''
```

Сплит-строка в Sql Server 2008/2012/2014 с использованием XML

Поскольку функция `STRING_SPLIT` отсутствует, нам нужно использовать XML-хак, чтобы разбить строку на строки:

Пример:

```
SELECT split.a.value('.', 'VARCHAR(100)') AS Value
FROM   (SELECT Cast ('<M>' + Replace('A|B|C', '|', '</M><M>')+ '</M>' AS XML) AS Data) AS A
       CROSS apply data.nodes ('/M') AS Split(a);
```

Результат:

```
+-----+
|Value|
+-----+
|A    |
+-----+
|B    |
+-----+
|C    |
+-----+
```

Переменная таблицы T-SQL и XML

```
Declare @userList Table(UserKey VARCHAR(60))
Insert into @userList values ('bill'),('jcom'),('others')
--Declared a table variable and insert 3 records

Declare @text XML
Select  @text = (
        select UserKey  from @userList for XML Path('user'), root('group')
```

```
)
--Set the XML value from Table

Select @text

--View the variable value
XML:
\<group>\<user>\<UserKey>bill\</UserKey>\</user>\<user>\<UserKey>jcom\</UserKey>\</user>\<user>\<UserKe
```

Прочитайте Функция Split String на сервере Sql онлайн: https://riptutorial.com/ru/sql-server/topic/3713/функция-split-string-на-сервере-sql

глава 108: Функция STUFF

параметры

параметр	подробности
character_expression	существующая строка в ваших данных
начальная_позиция	положение в `character_expression` выражении для удаления `length` а затем вставить `replacement_string`
длина	количество символов для удаления из `character_expression`
replacement_string	последовательность символов для вставки в `character_expression`

Examples

Замена основного символа с помощью STUFF ()

Функция `STUFF()` вставляет строку в другую строку, сначала удаляя указанное количество символов. В следующем примере удаляется «Svr» и заменяется на «Сервер». Это происходит, указывая `start_position` и `length` замены.

```
SELECT STUFF('SQL Svr Documentation', 5, 3, 'Server')
```

Выполнение этого примера приведет к возврату `SQL Server Documentation` вместо `SQL Svr Documentation`.

Использование FOR XML для объединения значений из нескольких строк

Одним из распространенных способов использования функции `FOR XML` является конкатенация значений нескольких строк.

Вот пример использования таблицы Customers :

```
SELECT
    STUFF( (SELECT ';' + Email
        FROM Customers
        where (Email is not null and Email <> '')
        ORDER BY Email ASC
        FOR XML PATH('')),
    1, 1, '')
```

В приведенном выше примере `FOR XML PATH('')` используется для конкатенации адресов

электронной почты, используя ; как символ разделителя. Кроме того, целью `STUFF` является удаление ведущего ; из конкатенированной строки. `STUFF` также неявно бросает конкатенированную строку из XML в varchar.

Примечание: результат из приведенного выше примера будет XML - кодировке, то есть он будет заменить < символы с `<` и т. д. Если вы этого не хотите, измените `FOR XML PATH(''))` в `FOR XML PATH, TYPE).value('.[1]','varchar(MAX)')` , например:

```
SELECT
    STUFF( (SELECT ';' + Email
        FROM Customers
        where (Email is not null and Email <> '')
        ORDER BY Email ASC
        FOR XML PATH, TYPE).value('.[1]','varchar(900)'),
    1, 1, '')
```

Это можно использовать для достижения результата, аналогичного `GROUP_CONCAT` в MySQL или `string_agg` в PostgreSQL 9.0+, хотя мы используем подзапросы вместо агрегатов GROUP BY. (В качестве альтернативы вы можете установить определяемый пользователем агрегат, такой как этот, если вы ищете функциональность, близкую к функциональности `GROUP_CONCAT`).

Получить имена столбцов, разделенные запятой (не список)

```
/*
The result can be use for fast way to use columns on Insertion/Updates.
Works with tables and views.

Example: eTableColumns  'Customers'
ColumnNames
----------------------------------------------------
Id, FName, LName, Email, PhoneNumber, PreferredContact

INSERT INTO Customers (Id, FName, LName, Email, PhoneNumber, PreferredContact)
    VALUES (5, 'Ringo', 'Star', 'two@beatles.now', NULL, 'EMAIL')
*/
CREATE PROCEDURE eTableColumns (@Table VARCHAR(100))
AS
SELECT ColumnNames =
   STUFF( (SELECT ', ' +  c.name
FROM
    sys.columns c
INNER JOIN
    sys.types t ON c.user_type_id = t.user_type_id
WHERE
    c.object_id = OBJECT_ID( @Table)
        FOR XML PATH, TYPE).value('.[1]','varchar(2000)'),
    1, 1, '')
GO
```

материал для запятой, разделенный на сервере sql

`FOR XML PATH` и `STUFF` объединить несколько строк в одну строку:

```
select distinct t1.id,
    STUFF(
           (SELECT ', ' + convert(varchar(10), t2.date, 120)
            FROM yourtable t2
            where t1.id = t2.id
            FOR XML PATH (''))
            , 1, 1, '')  AS date
  from yourtable t1;
```

Основной пример функции STUFF ().

STUFF (Original_Expression, Start, Length, Replacement_expression)

Функция STUFF () вставляет значение Replacement_expression в указанной начальной позиции вместе с удалением символов, заданных параметром Length.

```
Select FirstName, LastName,Email, STUFF(Email, 2, 3, '******') as StuffedEmail From Employee
```

Выполнение этого примера приведет к возврату данной таблицы

Имя	Фамилия	Эл. адрес	StuffedEmail
Jomes	охотник	James@hotmail.com	J*****s@hotmail.com
Shyam	rathod	Shyam@hotmail.com	S*****m@hotmail.com
Баран	Shinde	Ram@hotmail.com	R ***** hotmail.com

Прочитайте Функция STUFF онлайн: https://riptutorial.com/ru/sql-server/topic/703/функция-stuff

глава 109: Хранение JSON в таблицах SQL

Examples

JSON хранится в виде текстового столбца

JSON - текстовый формат, поэтому он хранится в стандартных столбцах NVARCHAR. Коллекция NoSQL эквивалентна таблице с двумя значениями столбцов:

```
CREATE TABLE ProductCollection (
  Id int identity primary key,
  Data nvarchar(max)
)
```

Используйте `nvarchar(max)` как вы не уверены, каков размер ваших документов JSON. `nvarchar(4000)` и `varchar(8000)` имеют лучшую производительность, но с ограничением по размеру до 8 КБ.

Убедитесь, что JSON правильно отформатирован с использованием ISJSON

Поскольку JSON хранит текстовый столбец, вы можете убедиться, что он правильно отформатирован. Вы можете добавить ограничение CHECK в столбец JSON, который проверяет, правильно ли форматируется текст JSON:

```
CREATE TABLE ProductCollection (
  Id int identity primary key,
  Data nvarchar(max)
       CONSTRAINT [Data should be formatted as JSON]
       CHECK (ISJSON(Data) > 0)
)
```

Если у вас уже есть таблица, вы можете добавить контрольное ограничение с помощью инструкции ALTER TABLE:

```
ALTER TABLE ProductCollection
    ADD CONSTRAINT [Data should be formatted as JSON]
        CHECK (ISJSON(Data) > 0)
```

Выводить значения из текста JSON в виде вычисленных столбцов

Вы можете вывести значения из столбца JSON в виде вычисленных столбцов:

```
CREATE TABLE ProductCollection (
  Id int identity primary key,
  Data nvarchar(max),
```

```
  Price AS JSON_VALUE(Data, '$.Price'),
  Color JSON_VALUE(Data, '$.Color') PERSISTED
)
```

Если вы добавите вычисленный столбец PERSISTED, значение из текста JSON будет реализовано в этом столбце. Таким образом, ваши запросы могут быстрее считывать значение из текста JSON, потому что не требуется синтаксический анализ. Каждый раз, когда JSON в этой строке изменяется, значение будет пересчитываться.

Добавление индекса по пути JSON

Запросы, которые фильтруют или сортируют данные по некоторому значению в столбце JSON, обычно используют полное сканирование таблицы.

```
SELECT * FROM ProductCollection
WHERE JSON_VALUE(Data, '$.Color') = 'Black'
```

Чтобы оптимизировать такие запросы, вы можете добавить неустановленный вычисленный столбец, который предоставляет выражение JSON, используемое в фильтре или сортировке (в этом примере JSON_VALUE (Data, «.Color»)) и создайте индекс в этом столбце:

```
ALTER TABLE ProductCollection
ADD vColor as JSON_VALUE(Data, '$.Color')

CREATE INDEX idx_JsonColor
ON ProductCollection(vColor)
```

Запросы будут использовать индекс вместо обычного сканирования таблицы.

JSON хранится в таблицах памяти

Если вы можете использовать таблицы с оптимизацией памяти, вы можете хранить JSON в виде текста:

```
CREATE TABLE ProductCollection (
  Id int identity primary key nonclustered,
  Data nvarchar(max)
) WITH (MEMORY_OPTIMIZED=ON)
```

Преимущества JSON в памяти:

- Данные JSON всегда находятся в памяти, поэтому нет доступа к диску
- При работе с JSON нет замков и защелок

Прочитайте Хранение JSON в таблицах SQL онлайн: https://riptutorial.com/ru/sql-server/topic/5029/хранение-json-в-таблицах-sql

глава 110: Хранимые процедуры

Вступление

В SQL Server процедура представляет собой хранимую программу, в которую вы можете передавать параметры. Он не возвращает значение, как функция. Тем не менее, он может вернуть статус успеха / отказа в процедуру, вызвавшую его.

Синтаксис

- СОЗДАТЬ {ПРОЦЕДУРА | PROC} [имя_схемы.] Имя_процесса
- [@parameter [type_schema_name.] тип данных
- [VARYING] [= по умолчанию] [OUT | ВЫХОД | READONLY]
- , @parameter [type_schema_name.] datatype
- [VARYING] [= по умолчанию] [OUT | ВЫХОД | READONLY]]
- [WITH {ENCRYPTION | РЕКОМЕНДУЕМ | ВЫПОЛНИТЬ КАК пункт}]
- [ДЛЯ РЕПЛИКАЦИИ]
- КАК
- НАЧАТЬ
- [Declaration_section]
- executable_section
- КОНЕЦ;

Examples

Создание и выполнение базовой хранимой процедуры

Использование таблицы « `Authors` в базе данных библиотеки

```
CREATE PROCEDURE GetName
(
    @input_id INT = NULL,       --Input parameter,  id of the person, NULL default
    @name VARCHAR(128) = NULL   --Input parameter, name of the person, NULL default
)
AS
BEGIN
    SELECT Name + ' is from ' + Country
    FROM Authors
    WHERE Id = @input_id OR Name = @name
END
GO
```

Вы можете выполнить процедуру с несколькими различными синтаксисами. Во-первых, вы можете использовать `EXECUTE` или `EXEC`

```
EXECUTE GetName @id = 1
EXEC Getname @name = 'Ernest Hemingway'
```

Кроме того, вы можете опустить команду EXEC. Кроме того, вам не нужно указывать, какой параметр вы передаете, когда вы передаете все параметры.

```
GetName NULL, 'Ernest Hemingway'
```

Если вы хотите указать входные параметры в другом порядке, чем то, как они объявлены в процедуре, вы можете указать имя параметра и присвоить значения. Например

```
 CREATE PROCEDURE dbo.sProcTemp
 (
    @Param1 INT,
    @Param2 INT
)
AS
BEGIN

    SELECT
        Param1 = @Param1,
        Param2 = @Param2

END
```

нормальный порядок выполнения этой процедуры - сначала указать значение для параметра @ Param1, а затем @ Param2 second. Так что это будет выглядеть примерно так

```
  EXEC dbo.sProcTemp @Param1 = 0,@Param2=1
```

Но также возможно, что вы можете использовать следующие

```
  EXEC dbo.sProcTemp @Param2 = 0,@Param1=1
```

в этом случае вы указываете значение для параметра @ param2 first и @ Param1 second. Это означает, что вам не нужно сохранять тот же порядок, что и в процедуре, но вы можете заказать любой заказ по своему усмотрению. но вам нужно указать, к какому параметру вы устанавливаете значение

Доступ к хранимой процедуре из любой базы данных

А также вы можете создать процедуру с префиксом `sp_` эти procuedres, как и все системные хранимые процедуры, могут быть выполнены без указания базы данных из-за поведения SQL Server по умолчанию. Когда вы выполняете хранимую процедуру, которая начинается с «sp_», SQL Server сначала ищет процедуру в основной базе данных. Если процедура не найдена в master, она просматривается в активной базе данных. Если у вас есть хранимая процедура, к которой вы хотите получить доступ из всех ваших баз данных, создайте ее в главном и используйте имя, которое включает префикс «sp_».

```
Use Master

CREATE PROCEDURE sp_GetName
(
    @input_id INT = NULL,       --Input parameter,  id of the person, NULL default
    @name VARCHAR(128) = NULL   --Input parameter, name of the person, NULL default
)
AS
BEGIN
    SELECT Name + ' is from ' + Country
    FROM Authors
    WHERE Id = @input_id OR Name = @name
END
GO
```

ЗАПОМНЕННАЯ ПРОЦЕДУРА с параметрами OUT

Хранимые процедуры могут возвращать значения, используя ключевое слово `OUTPUT` в списке параметров.

Создание хранимой процедуры с одним параметром out

```
CREATE PROCEDURE SprocWithOutParams
(
    @InParam VARCHAR(30),
    @OutParam VARCHAR(30) OUTPUT
)
AS
BEGIN
    SELECT @OutParam = @InParam + ' must come out'
    RETURN
END
GO
```

Выполнение хранимой процедуры

```
DECLARE @OutParam VARCHAR(30)
EXECUTE SprocWithOutParams 'what goes in', @OutParam OUTPUT
PRINT @OutParam
```

Создание хранимой процедуры с несколькими параметрами

```
CREATE PROCEDURE SprocWithOutParams2
```

```
(
    @InParam VARCHAR(30),
    @OutParam VARCHAR(30) OUTPUT,
    @OutParam2 VARCHAR(30) OUTPUT
)
AS
BEGIN
    SELECT @OutParam = @InParam +' must come out'
    SELECT @OutParam2 = @InParam +' must come out'
    RETURN
END
GO
```

Выполнение хранимой процедуры

```
DECLARE @OutParam VARCHAR(30)
DECLARE @OutParam2 VARCHAR(30)
EXECUTE SprocWithOutParams2 'what goes in', @OutParam OUTPUT, @OutParam2 OUTPUT
PRINT @OutParam
PRINT @OutParam2
```

Сохраненная процедура с If ... Else и Insert Into operation

Создать таблицу примеров `Employee` :

```
CREATE TABLE Employee
(
    Id INT,
    EmpName VARCHAR(25),
    EmpGender VARCHAR(6),
    EmpDeptId INT
)
```

Создает хранимую процедуру, которая проверяет, не являются ли значения, переданные в хранимой процедуре, нулевыми или не пустыми и выполняют операцию вставки в таблице Employee.

```
CREATE PROCEDURE spSetEmployeeDetails
(
    @ID int,
    @Name VARCHAR(25),
    @Gender VARCHAR(6),
    @DeptId INT
)
AS
BEGIN
    IF (
        (@ID IS NOT NULL AND LEN(@ID) !=0)
        AND (@Name IS NOT NULL AND LEN(@Name) !=0)
        AND (@Gender IS NOT NULL AND LEN(@Gender) !=0)
        AND (@DeptId IS NOT NULL AND LEN(@DeptId) !=0)
    )
    BEGIN
```

```
        INSERT INTO Employee
        (
            Id,
            EmpName,
            EmpGender,
            EmpDeptId
        )
        VALUES
        (
            @ID,
            @Name,
            @Gender,
            @DeptId
        )
    END
ELSE
    PRINT 'Incorrect Parameters'
END
GO
```

Выполнение хранимой процедуры

```
DECLARE @ID INT,
    @Name VARCHAR(25),
    @Gender VARCHAR(6),
    @DeptId INT

EXECUTE spSetEmployeeDetails
    @ID = 1,
    @Name = 'Subin Nepal',
    @Gender = 'Male',
    @DeptId = 182666
```

Динамический SQL в хранимой процедуре

Dynamic SQL позволяет создавать и запускать SQL-запросы во время выполнения. Динамический SQL необходим, когда наши операторы SQL содержат идентификатор, который может меняться в разное время компиляции.

Простой пример динамического SQL:

```
CREATE PROC sp_dynamicSQL
@table_name      NVARCHAR(20),
@col_name        NVARCHAR(20),
@col_value       NVARCHAR(20)
AS
BEGIN
DECLARE  @Query  NVARCHAR(max)
SET      @Query = 'SELECT * FROM ' + @table_name
SET      @Query = @Query + ' WHERE ' + @col_name + ' = ' + ''''+@col_value+''''
EXEC     (@Query)
END
```

В приведенном выше sql-запросе мы можем видеть, что мы можем использовать вышеуказанный запрос, определяя значения в `@table_name, @col_name, and @col_value` во

время выполнения. Запрос генерируется во время выполнения и выполняется. Это метод, в котором мы можем создавать целые скрипты в виде строки в переменной и выполнять ее. Мы можем создавать более сложные запросы с использованием динамической концепции SQL и конкатенации. Эта концепция очень эффективна, если вы хотите создать сценарий, который можно использовать в нескольких условиях.

Выполнение хранимой процедуры

```
DECLARE @table_name     NVARCHAR(20) = 'ITCompanyInNepal',
        @col_name       NVARCHAR(20) = 'Headquarter',
        @col_value      NVARCHAR(20) = 'USA'

EXEC    sp_dynamicSQL   @table_name,
                        @col_name,
                        @col_value
```

Таблица, которую я использовал

ID	CompanyName	CompanyAddress	Headquarter	NumberOfEmployee
1	CompanyOne	Kathmandu	USA	300
2	CompanyTwo	Kathmandu	USA	260
3	CompanyThree	Kathmandu	Nepal	300
4	CompanyFour	Kathmandu	Nepal	180
6	CompanySix	Janakpur	USA	50
7	CompanySeven	Janakpur	Australia	100
8	CompanyEight	Birganj	Australia	150
9	CompanyNine	Biratnagar	Canada	200
10	CompanyTen	Pokhara	India	85

Выход

ID	CompanyName	CompanyAddress	Headquarter	NumberOfEmployee
1	CompanyOne	Kathmandu	USA	300
2	CompanyTwo	Kathmandu	USA	260
6	CompanySix	Janakpur	USA	50
1	CompanyA	Banglore	USA	400
2	CompanyB	Banglore	USA	450

Простая петля

Сначала можно получить некоторые данные в таблице temp с именем `#systables` и `#systables` число увеличивающихся строк, чтобы мы могли запросить одну запись за раз

```
select
    o.name,
    row_number() over (order by o.name) as rn
into
    #systables
from
    sys.objects as o
where
```

```
    o.type = 'S'
```

Далее мы объявляем некоторые переменные для управления циклом и хранения имени таблицы в этом примере

```
declare
    @rn int = 1,
    @maxRn int = (
                    select
                        max(rn)
                    from
                        #systables as s
                    )
declare    @tablename sys name
```

Теперь мы можем циклично использовать простую. Мы увеличиваем `@rn` в выражении `select` но это также может быть отдельный оператор ex `set @rn = @rn + 1` который будет зависеть от ваших требований. Мы также используем значение `@rn` до того, как оно будет увеличено, чтобы выбрать одну запись из `#systables` . Наконец, мы печатаем имя таблицы.

```
while @rn <= @maxRn
    begin

        select
            @tablename = name,
            @rn = @rn + 1
        from
            #systables as s
        where
            s.rn = @rn

        print @tablename
    end
```

Простая петля

```
CREATE PROCEDURE SprocWithSimpleLoop
(
    @SayThis VARCHAR(30),
    @ThisManyTimes INT
)
AS
BEGIN
    WHILE @ThisManyTimes > 0
    BEGIN
        PRINT @SayThis;
        SET @ThisManyTimes = @ThisManyTimes - 1;
    END

    RETURN;
END
GO
```

Прочитайте Хранимые процедуры онлайн: https://riptutorial.com/ru/sql-server/topic/3213/

хранимые-процедуры

глава 111: Цикл WHILE

замечания

Использование цикла `WHILE` или другого итеративного процесса обычно не является наиболее эффективным способом обработки данных в SQL Server.

Вы должны предпочесть использовать набор данных на основе данных для достижения тех же результатов, где это возможно

Examples

Использование цикла While

Цикл `WHILE` может использоваться как альтернатива `CURSORS` . В следующем примере будут напечатаны цифры от 0 до 99.

```
DECLARE @i int = 0;
WHILE(@i < 100)
BEGIN
   PRINT @i;
   SET @i = @i+1
END
```

В то время как цикл с использованием агрегата min min

```
DECLARE @ID AS INT;

SET @ID = (SELECT MIN(ID) from TABLE);

WHILE @ID IS NOT NULL
BEGIN
    PRINT @ID;
    SET @ID = (SELECT MIN(ID) FROM TABLE WHERE ID > @ID);
END
```

Прочитайте Цикл WHILE онлайн: https://riptutorial.com/ru/sql-server/topic/4249/цикл-while

глава 112: шифрование

параметры

Дополнительные параметры	подробности
`WITH PRIVATE KEY`	Для CREATE CERTIFICATE может быть указан закрытый ключ: `(FILE='D:\Temp\CertTest\private.pvk', DECRYPTION BY PASSWORD = 'password');`

замечания

Создание сертификата DER будет работать нормально. Однако, когда используется сертификат Base64, SQL-сервер будет жаловаться на критическое сообщение:

```
Msg 15468, Level 16, State 6, Line 1
An error occurred during the generation of the certificate.
```

Импортируйте свой сертификат Base64 в хранилище сертификатов вашей ОС, чтобы иметь возможность реэкспортировать его в двоичный формат DER.

Еще одна важная вещь - иметь иерархию шифрования, чтобы одна защищала другую, вплоть до уровня ОС. См. Статью «Шифрование базы данных / TDE»

Для получения дополнительной информации для создания сертификатов перейдите по адресу : https://msdn.microsoft.com/en-us/library/ms187798.aspx

Для получения дополнительной информации о шифровании базы данных / TDE перейдите по адресу : https://msdn.microsoft.com/en-us/library/bb934049.aspx

Для получения дополнительной информации для шифрования данных перейдите по адресу : https://msdn.microsoft.com/en-us/library/ms188061.aspx

Examples

Шифрование по сертификату

```
CREATE CERTIFICATE My_New_Cert
FROM FILE = 'D:\Temp\CertTest\certificateDER.cer'
GO
```

Создать сертификат

```
SELECT EncryptByCert(Cert_ID('My_New_Cert'),
'This text will get encrypted') encryption_test
```

Обычно вы шифруете симметричным ключом, этот ключ будет зашифрован с помощью асимметричного ключа (открытого ключа) из вашего сертификата.

Также обратите внимание, что шифрование ограничено определенной длиной в зависимости от длины ключа и возвращает NULL в противном случае. Microsoft пишет: «Предельные значения: 512-битный ключ RSA может шифровать до 53 байтов, 1024-битный ключ может шифровать до 117 байтов, а ключ 2048 бит может шифровать до 245 байт».

EncryptByAsymKey имеет те же ограничения. Для UNICODE это будет разделено на 2 (16 бит на символ), поэтому 58 символов для 1024-битного ключа.

Шифрование базы данных

```
USE TDE
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE My_New_Cert
GO

ALTER DATABASE TDE
SET ENCRYPTION ON
GO
```

Это использует «прозрачное шифрование данных» (TDE)

Шифрование с помощью симметричного ключа

```
-- Create the key and protect it with the cert
CREATE SYMMETRIC KEY My_Sym_Key
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE My_New_Cert;
GO

-- open the key
OPEN SYMMETRIC KEY My_Sym_Key
DECRYPTION BY CERTIFICATE My_New_Cert;

-- Encrypt
SELECT EncryptByKey(Key_GUID('SSN_Key_01'), 'This text will get encrypted');
```

Шифрование парольной фразой

```
SELECT EncryptByPassphrase('MyPassPhrase', 'This text will get encrypted')
```

Это также зашифровывает, а затем парольная фраза вместо асимметричного ключа (сертификата) или явного симметричного ключа.

Прочитайте шифрование онлайн: https://riptutorial.com/ru/sql-server/topic/7096/шифрование

глава 113: Экспорт данных в txt-файл с помощью SQLCMD

Синтаксис

- sqlcmd -S SHERAZM-E7450 \ SQL2008R2 -d Baseline_DB_Aug_2016 -o c: \ employee.txt -Q "выберите * у сотрудника"

Examples

Используя SQLCMD в командной строке

Структура команд

sqlcmd -S yourservername \ instancename -d имя_базы базы данных -o outputfilename_withpath -Q "ваш запрос выбора"

Переключатели следующие

-S для имени сервера и имени экземпляра

-d для исходной базы данных

-o для целевого файла вывода (он будет создавать выходной файл)

-Q для запроса на выборку данных

Прочитайте Экспорт данных в txt-файл с помощью SQLCMD онлайн: https://riptutorial.com/ru/sql-server/topic/7076/экспорт-данных-в-txt-файл-с-помощью-sqlcmd

кредиты

S. No	Главы	Contributors
1	Начало работы с Microsoft SQL Server	Abhilash R Vankayala, Abhishek Jain, Ahmad Aghazadeh, Ahmar, Akshay Anand, alalp, Almir Vuk, Arthur D, ATC, Athafoud, BeaglesEnd, Bhanu, Biju jose, Blachshma, bluefeet, ChrisM, Christos, Community, cteski, D M, Darshak, Gidil, Gordon Bell, Greg Bray, Iztoksson, Jared Hooper, JerryOL, Job AJ, Joe Taras, John Odom, John Slegers, JonasCz, K48, kafka, Lamak, Laughing Vergil, Mahesh Dahal, Malt, Martin Smith, Matt, Matt, Max, Mihai-Daniel Virna, Mudassir Hasan, n00b, Nick, Nikolay Kostov, onupdatecascade, OzrenTkalcecKrznaric, Peter Tirrell, Phrancis, Prateek, Sam, Shaneis, Thuta Aung, Tony L., Tot Zam, Uberzen1, Umachandar - Microsoft, user_0, user2314737, VoidDemon, Zsuzsa
2	bcp (программа массовой копии) Утилита	MarmiK
3	COALESCE	Bharat Prasad Satyal, Edathadan Chief aka Arun, Karthikeyan, Matej, scsimon, Tab Alleman
4	DBCC	Jovan MSFT
5	DBMAIL	Phrancis
6	FileStream	Raghu Ariga
7	FOR XML PATH	bluefeet, gotqn, Keith Hall, Wolfgang
8	JSON на сервере Sql	Jovan MSFT, Mono
9	MERGE	Abhilash R Vankayala, Abubakar Riaz, Alex, David Kaminski, dd4711, Hari K M, Moshiour, Rogerio Soares, Serg
10	OPENJSON	James, Jovan MSFT
11	ParseName	Mani
12	PHANTOM читать	Max
13	PIVOT / UNPIVOT	Athafoud, bluefeet, DhruvJoshi, kolunar

14	Schemas	Merenix
15	SCOPE_IDENTITY ()	Dheeraj Kumar
16	SQL Server Evolution через различные версии (2000 - 2016)	Dan Guzman, M.Ali
17	SQLCMD	Eugene Niemand, Techie
18	UNION	cnayak
19	Анализ запроса	DForck42
20	Безопасность на уровне строк	Carsten Hynne, Jovan MSFT
21	Временные таблицы	Ahmad Aghazadeh, Akshay Anand, Ben O, Mspaja
22	Вставить	Randall
23	ВСТАВИТЬ В	Abubakar Riaz, barcanoj, DVJex, Hari K M, intox, martinshort, Matas Vaitkevicius, Max, Michael Stum, n00b, Piotr Nawrot, Robert Columbia, Tot Zam, woony
24	Встроенная память OLTP (Hekaton)	Akshay Anand, Behzad, Brandon, Jovan MSFT, Martijn Pieters
25	Вычисляемые столбцы	cnayak, Kannan Kandasamy
26	ГРУППА ПО	Andy, Edathadan Chief aka Arun, Jenism, juergen d, Julien Vavasseur, Kiran Ukande, Matas Vaitkevicius, ShlomiR
27	Группа файлов	Behzad
28	Губернатор ресурсов	Ako, RamenChef
29	Даты	A_Arnold, Adam Porad, Akshay Anand, Bellash, cteski, Edathadan Chief aka Arun, JamieA, Jared Hooper, Kritner, Lamak, Mert Gülsoy, Nick, Phrancis, SHD, Siyual, Soukai, UnhandledExcepSean, Zohar Peled
30	Динамический SQL	Jovan MSFT
31	Динамический SQL Pivot	Jesse
32	Динамическое маскирование данных	Jovan MSFT

33	ДЛЯ JSON	James, Jovan MSFT
34	ЕСЛИ ЕЩЕ	cteski, M.Ali, RamenChef
35	ЗАВЕРШЕНИЕ	Athafoud, bluefeet, Brandon, DVT, gofr1, Lamak, Paul Bambury, RamenChef, Rowland Shaw, Sam
36	Запланированная задача или задание	Edathadan Chief aka Arun, Hadi
37	Запрос результатов по странице	Pat
38	Запросы с данными JSON	bakedpatato, James, Jovan MSFT
39	Заявление CASE	Laughing Vergil, RamenChef, Vikas Vaidya
40	Значения NULL	Amir Pourmand, Hadi, Kannan Kandasamy, Kritner, Laughing Vergil, podiluska, Sean Branchaw, Zohar Peled
41	Изменить текст JSON	Jovan MSFT
42	Имена псевдонимов на сервере Sql	Pரதீ ப
43	Импорт BULK	Jovan MSFT
44	Индекс	Ahmad Aghazadeh, Akshay Anand, cteski, Henrik Staun Poulsen, Martin Smith, Tom V
45	Иностранные ключи	Jovan MSFT
46	Интеграция с обычным языком	Jovan MSFT
47	Использование таблицы TEMP	APH, New
48	Исходно скомпилированные модули (Hekaton)	bakedpatato, Jovan MSFT
49	Клавиши быстрого доступа Microsoft SQL Server Management Studio	Bino Mathew Varghese, cteski, Sibeesh Venu

50	КЛАСТЕРИРОВАННАЯ КОЛОНКА	Jovan MSFT
51	Ключевое слово Drop	Ignas, Jakub Ojmucianski, Justin Rohr, Max, scsimon
52	курсоры	Kane, Phrancis
53	Логические функции	dd4711
54	Магазин запросов	bakedpatato, Jovan MSFT
55	миграция	Matas Vaitkevicius
56	Обозначение специальных символов и зарезервированных слов	bassrek
57	Обработка транзакций	Metanormal
58	Общие выражения таблицы	Arif, bbrown, cteski, DForck42, Jeffrey Van Laethem, Jovan MSFT, kafka, Keith Hall, Monty Wild, SQLMason
59	Ограничить набор результатов	alalp, chrisb, cteski, ErikE
60	Оператор SELECT	cteski, Jovan MSFT
61	Опция Ties	TheGameiswar
62	Основные операции DDL в MS SQL Server	Matt
63	пагинация	cteski, Jovan MSFT, Sender
64	Первичные ключи	Kritner
65	переменные	APH, Keith Hall, Phrancis
66	Перемещение и копирование данных вокруг таблиц	Nick.McDermaid
67	подзапросов	cnayak
68	Подсказки по запросам	cteski, DARKOCEAN, Jovan MSFT, user_0

69	Полнотекстовая индексация	Edathadan Chief aka Arun
70	Получить информацию о базе данных	Andrea, Anuj Tripathi, Baodad, Brent Ozar, dario, feetwet, James Anderson, JamieA, Jasmin Solanki, Jeffrey Van Laethem, jyao, Kritner, Laughing Vergil, LowlyDBA, Mahendra, Moshiour, Phrancis, Rhumborl, scsimon, Shiva, spaghettidba, Tot Zam, TZHX, Umachandar - Microsoft
71	Получить информацию о вашем экземпляре	Bino Mathew Varghese, feetwet, James Anderson, Kritner, LowlyDBA, S.Karras, scsimon
72	ПОПРОБУЙ ПОЙМАТЬ	Jovan MSFT, ravindra, Uberzen1
73	Последняя введенная идентификация	Jeffrey Van Laethem, sqluser, Tot Zam
74	Последовательности	Josh Morel
75	Преобразование типов данных	Ben O, Edathadan Chief aka Arun
76	Привилегии или разрешения	Oluwafemi
77	применять крест	Hamza Rabah, Jovan MSFT, Tom V
78	Присоединиться	4444, Akshay Anand, Andy, APH, Bino Mathew Varghese, cteski, Dean Ward, DhruvJoshi, Dileep, Gajendra, HK1, Iztoksson, Jeffrey Van Laethem, Joao Araujo, JonH, L J, Lamak, Laughing Vergil, LowlyDBA, mtb, Nikolay Kostov, OzrenTkalcecKrznaric, Phrancis, Ram Grandhi, SqlZim
79	Просмотры	Benjamin Hodgson, Daniel Lemke, Max
80	Пространственные данные	cteski, Neil Kennedy, RamenChef, Vladimir Oselsky
81	Разметка	Dan Guzman, James Anderson, John Odom, Susang
82	Разрешения базы данных	Ben Thul
83	Разрешения и безопасность	5arx
84	Расширенные настройки	Ahmad Aghazadeh

85	Резервное копирование и восстановление базы данных	Jones Joseph, Jovan MSFT
86	Сервисный брокер	Ken S., Matej, RamenChef
87	Системная база данных - TempDb	Anuj Tripathi, RamenChef
88	Снимки базы данных	Akash, Daryl, Jovan MSFT, Wolfgang
89	Совокупные функции	Akshay Anand, cnayak, cteski, Jeffrey L Whitledge, Joe Taras, Vexator
90	Создание диапазона дат	James, Siyual
91	СОЗДАТЬ ВИД	Almir Vuk, cteski, Edathadan Chief aka Arun, Hadi, Josh B, Robert Columbia, Tot Zam
92	СОРТИРОВАТЬ ПО	APH, beercohol, cteski, Gidil, RamenChef
93	Сортировка / упорядочение строк	APH, OzrenTkalcecKrznaric
94	Спусковой крючок	Oluwafemi, The_Outsider, Zohar Peled
95	Строковые функции	A_Arnold, anon, cteski, FoxyBOA, Hadi, hatchet, Igor Micev , Jibin Balachandran, Jovan MSFT, mtb, Phrancis, Raidri, Ricardo C, Ross Presser, takrl, Zohar Peled
96	Студия управления SQL Server (SSMS)	dd4711
97	Табличные параметры	Zohar Peled
98	Типы данных	Laughing Vergil, Matas Vaitkevicius
99	Типы пользовательских таблиц	Jivan, Zohar Peled
100	Управление базами данных Azure SQL	Jovan MSFT
101	Уровни изоляции и блокировка	RamenChef, sqlandmore.com

102	Уровни изоляции транзакций	Phrancis
103	Установка SQL Server в Windows	Luis Bosquez
104	Функции агрегации строк в SQL Server	Kannan Kandasamy
105	Функции окна	andyabel, feetwet, MarmiK
106	Функции ранжирования	cteski, kolunar, New
107	Функция Split String на сервере Sql	Jibin Balachandran, Jovan MSFT, MasterBob, Pரதீ ப், RamenChef
108	Функция STUFF	Arthur D, bluefeet, Chetan Sanghani, dacohenii, Kiran Ukande, Luis Bosquez, MrE, user1690166
109	Хранение JSON в таблицах SQL	Ed Harper, Jovan MSFT, RamenChef
110	Хранимые процедуры	Bino Mathew Varghese, cnayak, cteski, Erik Oppedijk, Eugene Niemand, Hari K M, Jayasurya Satheesh, Matas Vaitkevicius, Nathan Skerl, Pirate X, scsimon
111	Цикл WHILE	lord5et, Matas Vaitkevicius, podiluska, RamenChef, Wojciech Kazior
112	шифрование	Rubenisme
113	Экспорт данных в txt-файл с помощью SQLCMD	sheraz mirza